VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2013 Bc. David Novotný

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky (460)

Sociální síť pro tvorbu kvízových otázek

Social Network for Creation of Quiz Questions

2013 Bc. David Novotný

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. David Novotný**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Sociální síť pro tvorbu kvízových otázek
Social Network for a Creation of Quiz Questions

Zásady pro vypracování:

Cílem práce je vytvoření sociální sítě pro zadávání kvízových otázek a odpovědí (špatných i správných), které pak budou dostupné přes API pro kvízové aplikace třetích stran. Jednotliví uživatelé, kteří by chtěli zadávat otázky a odpovědi by se tedy registrovali. U jednotlivých otázek a odpovědí bude možné:
1. Zadat více správných odpovědí.
2. Otázky by mohly obsahovat obrázky a případně výrazy (jako matematika) vyjádřené pomocí XML.
3. Otázky zařazovat do kategorií.
4. Otázky a odpovědi diskutovat jednotlivými uživateli a přiřadit k nim doprovodný text s odkazy.
5. Každý uživatel by mohl sledovat otázky v určitých kategoriích, aby byl upozorněn na případné nové otázky, či změny v otázkách/odpovědích.

Vždy bude existovat jeden tvůrce otázky/odpovědí, nicméně další uživatelé budou moci navrhovat opravy (kromě samotné diskuze) a tyto návrhy bude možné s komentářem přijmout či zamítnout. Dále by aplikace obsahovala správu jednotlivých kategorií, aby bylo možné kategorie členit do hierarchického stromu (jako obsah v knize). Mělo by být možné celé kategorie a jejich podstromy spojovat.

Dále bude vytvořeno API (například formou webové služby), které by tyto otázky a kategorie zpřístupňovalo dalším aplikacím.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 6.5.2013 .....

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé diplomové práce Ing. Radimu Bačovi, Ph.D. za rady a také jeho čas, který mi věnoval při četných konzultacích. Dále děkuji rodině za podporu během studia a přítelkyni Aničce za její trpělivost a gramatickou korekturu.

Abstrakt

V této práci se zabýváme implementací sociální sítě pro tvorbů kvízových otázek. Hlavním cílem je vytvořit webovou aplikaci, která bude uživatelům umožňovat jednoduchou tvorbu kvízů. Ty bude možno diskutovat a take opravovat prostřednictvím návrhů ostatních uživatelů. Dalším cílem sociální sítě je vytvořit funkcionalitu pro testování kvízů a jejich řazení do domácích úkolů. Zaměřujeme se také na využití našich dat aplikacemi třetích stran, pro které vytvoříme API a také aplikaci jako ukázku jeho použití. Součástí práce je i popis teoretických základů. Naše řešení budeme implementovat za využití technologií rodiny .NET, konkrétně tedy ASP.NET, WCF, SignalR a pro databáze pak Microsoft SQL Server a Redis.

Klíčová slova

Kvíz, Sociální síť, ASP.NET, SignalR, WCF, MS SQL Server, Redis

Abstract

In this work we deal with implementation of social network for a creation of quiz questions. The main objective is the create website application, which allow users to simply create quizzes. Users will be able to discuss and also suggest repair of quizzes. Another goal of social network is to create a functionality for testing quizzes and sort them into homeworks. We will also provide our data to third party applications for that reason will be created API and also desktop application as an example of its use. Part of work is also a description of the theoretical foundations. For implementation of our solution we will use .NET technology. Specifically ASP.NET, WCF, SignalR and for the databases we will use Microsoft SQL Server and Redis.

Key words

Quiz, Social network, ASP.NET, SignalR, WCF, MS SQL Server, Redis

Obsah

Kapitola 1

1. Úvod

V dnešní době, kdy je přístupem na internet vybavená většina lidí v civilizovaných zemích, se projevuje jejich potřeba sdružovat se do různých skupin. Tyto komunity mohou vzniknout například na základě společných zájmů, potřebě mezi sebou komunikovat nebo sdílet různý obsah. Z pohledu vývoje internetu je právě sdílení obsahu hlavním faktorem změny základního pohledu na šíření informací na internetu. Uživatel informace již pouze nevyhledává, ale sám se podílí na jejich tvorbě. Místem, kde se lidé na internetu sdružují, je převážně webová aplikace. Ta je obvykle dostupná po registraci uživatatele a následně mu umožňuje využívat její funkce, kterými může být již zmíněná tvorba obsahu. Dnes je velice populární takovéto webové aplikace označovat jako sociální sítě. Pod pojmem sociální síť si dnes většina lidé představí Facebook, ale takto lze označit také různá fóra, chaty a další aplikace koncepce Webu 2.0. Dalším nezanedbatelným prostředkem pro využívání sociálních sítí jsou klienti různých platforem. Důvodem je stale se zvyšující počet přístupů k sociálním sítím jejich prostřednictvím. Toto může být následkem rozšiřování dostupnosti internetového pokrytí a s ním spojeného využívání moderních zařízení jako jsou například tablety a chytré telefony.

S ohledem k naší potřebě vytvářet kvízové otázky a sdílet je s dalšími lidmi jsme nejdříve hledali aplikaci s požadovanou funkčností. Tou rozumíme hlavně jednoduché vytváření kvízových otázek s důrazem na rozšířenou funkcionalitu jejich tvorby. Dále bylo důležité i oborové zaměření aplikace. Na základě rešerše popsané v kapitole 2. Přehled stávajících řešení jsme zjistili, že žádné z existujících řešeních plně nevyhovuje našim požadavkům. Ať už se jedná například o specifický obor kvízů zamýšlený pro budoucí využití nebo take nějaký typ automatického asistenta při vytváření kvízu využívajícího například sémantických dat. Proto jsme dospěli k názoru, že zde exsituje místo pro přetvoření našich požadavků do úplně nové aplikace.

Cílem diplomové práce je tedy vytvořit webovou aplikaci, která bude uživatelům umožňovat jednoduchou tvorbu kvízů. Ty bude možno diskutovat a take opravovat prostřednictvím návrhů ostatních uživatelů. Dalším cílem sociální sítě je vytvořit funkcionalitu pro testování kvízů a jejich řazení do domácích úkolů. Zaměřujeme se také na využití našich dat aplikacemi třetích stran, pro které vytvoříme API a také aplikaci jako ukázku jeho použití. Součástí práce je i popis teoretických základů.

Kapitola 2

2. Přehled stávajících řešení

Pro přehlednost je nejprve nutné vymezit hlavní požadavky na webovou aplikaci, na jejichž základě jsme přistoupili k hodnocení stávajících řešení. Těmi jsou například možňost vytváření kvízových otázek a jejich členění do kategorií. Kategorie by měly tvořit editovatelnou hierarchii. Při vytváření otázek by měl existovat určitý typ podpory jejich tvorby hlavně pro oblast databázové terminologie a SQL dotazů. Na tuto oblast by měla být také zaměřena nezanedbatelná část obsahu již hotových otázek. Dalším požadavkem je existence prvků moderní sociální sítě fungující v reálném čase jako je upozorňovací systém nebo možnost diskuze. V neposlední řadě také požadujeme, aby uživatel mohl navrhnout opravu otázky. Všechny hlavní funkční požadavky jsou posány v kapitole 4.1 Analýza a návrh, konkrétně v tabulkách 4.1, 4.3 a 4.4.

Cílem rešerše je tedy zmapovat existující softwarové řešení pro tvorbu kvízových otázek s výše popsanými funkcemi. Během hledání jsme narazili na problém velkého počtu nekvalitních řešení. Nekvalitním řešením máme na mysli aplikace a stránky, které nesplňovaly naše požadavky co se týče rozsahu jak dat obsažených v databázi tak omezenou funkčností. Mnoho z těchto řešení se zabývá malou oblastí s nízkou informační hodnotou, která souvisí se zábavným zaměřením webů, jejichž jsou součástí. Dalším příznakem nízké kvality byla častá replikace obsahu. Díky velkému množství těchto výsledků bylo mezi nimi obtížné najít řešení vyhovující našemu cíli. Proto jsme do výsledků zařadili i systémy LMS (Learning Management Systems).

2.1 Google Docs (Form)

Google Docs nabízí možnost vytvořit Form, který může obsahovat libovolný počet otázek. Existuje zde sedm druhů otázek. Vytváření Formu je intuitivní a uživatelsky přívětivé. K tomu, abychom mohli Google Docs používat, potřebujeme Google účet. Tady se dostáváme k sociláním funkcím tohoto řešení. Díky Google účtu může být vytvořený Form sdílen na sociální síťi Google+. Pokud nevlastníme účet na této síti, můžeme Form sdílet prostřednictvím emailu. Ani v jednom z těchto případů se nejedná o plnohodnotnou sociální síť pro tvorbu kvízových otázek, ovšem určitý druh sociálních funkcí obsahuje. Příjemnou funkcí jsou statistiky, které obsahují různé grafy poměrů odpovědí nebo historii odpovědí v čase. Podle našeho názoru jde o nejzákladnější prostředí pro tvorbu kvízových otázek s omezenými sociálními funkcemi a analýzou odpovědí.

2.2 ProProfs

Jedná se o webovou stránku, kterou lze zařadit do kategorie řešení s vyšší kvalitou. V tomto případě je kvalita svázána s dostupností. Jinými slovy stránka poskytuje zdarma pouze omezenou funkčnost a za měsíční poplatek nabízí rozšířené možnosti. Důvodem proč zmiňujeme tuto stránku je, že se nám líbí zpracování její zdarma dostupné verze. Konkrétně jde o vytváření otázek do kvízu. Tato stránka umožňuje přidávání otázek do kvízu nejen ručně jejich vyplněním, ale také nabízí funkci vyhledávání podle klíčových slov v databázi již vytvořených otázek. Tato funkce je ojedinělá a proto stojí za zmínku. Podobně si představujeme uživatelský pohled na naši plánovanou funkci automatického návrhu otázek. Co se týče sociálních funkcí, ty jsou dostupné pouze v placené verzi stránek a opět pouze možnost sdílení na již existující sociální sítě Facebook a Twitter nebo prostřednictvím emailu.

2.3 TestQ

Testq je zástupcem kategorie kvízů, které jsou součástí jakéhosi magazínu. Tato stránka po registraci nabízí funkce asi nejbližší sociální síti. Je zde klient pro zasílání i příjímání zpráv a dokonce i "zeď", na které se objevuje poslední aktivita registrovaných členů. Bohužel toto řešení neobsahuje základní námi požadovanou funkci a to je tvorba otázek. V tomto případě je jejich tvorba na administrátorech a z registrovaných členů jsou pouze konzumenti obsahu. Ti navíc mezi sebou soutěží ve vypracovávání testů a zisk bodů podle úspěšnosti odpovězení se pak promítá na žebříčku.

2.4 ZohoChallange

Tato stránka je podobná již zmiňovanému ProProfs. Také nabízí placenou a neplacenou verzi. Při vytváření otázek máme však na výběr pouze ze čtyř typů. Její výhodou pak může být její distribuce i jako doplňek do prohlížeče Chrome. Tak jako všechny ostatní je určen pro tvorbu online testů.

Stejně tak jako ProProfs bohužel nedosahuje potřebných kvalit popsaných v úvodu kapitoly a to díky absenci systému podpory tvorby kvízových otázek tak jako upozorňovací systém a diskuze v reálném čase a další. Také námi preferovaná oblast zaměření otázek zde nemá zastoupení.

2.5 LMS Systémy

LMS jsou aplikace, které obsahují různé online nástroje pro komunikaci a řízení studia v rámci e-learningu. Těchto aplikací existuje celá řada, od těch nejjednodušších až po složité. Jako příklad za všechny uvedeme systém Moodle, který používá i VŠB-TUO. Moodle je vyvíjen jako open source software a je tedy zdarma. Celý systém se skládá z mnoha modulů, které tvůrci dovolují do online kurzu vkládat výukové materiály jako HTML stránky, soubory ke stažení nebo Flash videa. Dále pak můžeme vkládat diskuzní fóra, automaticky vyhodnocované testy, ankety a další obsah dle specifikace SCROM nebo IMS Content Package. Moodle zaznamenává činnost uživatelů do protokolů a souhrnných statistik. Pro rozšíření jeho funkcionality je možné jej napojit na další systémy jako LDAP, IMAP nebo Mahara. Poslední zmiňovaný slouží jako webové portfolio nebo sociální síť. Co se týče velikosti projektu i jeho použitelnosti patří Moodle mezi nejlepší.

Přes svoji propracovanost však Moodle nesplňuje některé z našich požadavků. Těmi jsou například návrh opravy nebo notifikační systém a diskuze v reálném čase. Také neobsahuje žádnou podporu tvorby kvízových otázek pro náš tématický okruh. Při případném použití tohoto systému bychom začínali od začátku bez jakýchkoli dat v databázi. Tím zaniká i další potencionální výhoda oproti našemu systému, který by na tom byl s množstvím dat na začátku stejně. Narozdíl od Moodlu by v naší aplikaci jejich tvorba měla být jednodušší, což je další důvod pro nepoužití ani tohoto kandidáta.

2.6 Zhodnocení výsledků

Naším cílem bylo získat povědomí o schopnostech potenciálně konkurenčních řešeních v oblasti sociálních sítí pro tvorbu kvízových otázek. Tento cíl se nám podařilo zvládnout a jeho výsledky prokazují, že navzdory výborným řešením zvláště v oblasti LMS systémů, se nabízí prostor pro vývoj našeho softwaru. Jako hlavní výhodu oproti konkurentům vidíme ve funkci asistenta při vytváření nových otázek a také zaměření se na specifickou tématickou oblast IT technologií. Dále pak může být výhodou zaměření se na vývoj projektu jako sociální síťe. Během průzkumu jsme totiž narazili na absenci řešení s podobnými ambicemi.

Kapitola 3

3. Teoretický úvod

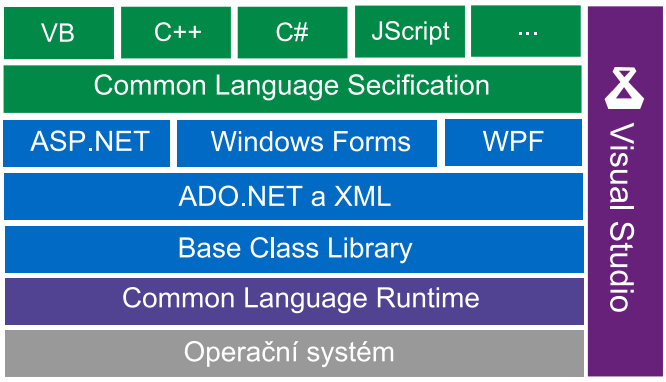
3.1 Využité technologie

V rámci diplomové práce řešíme vývoj webové aplikace, API a ukázkové desktopové aplikace. Jedná se o více aplikací a tak jedním z faktorů při výběru technologií byla implementační ucelenost. Proto byla pro implementaci tohoto projektu vybrána technologie firmy Microsoft. Našim účelům plně vyhovuje díky tomu, že můžeme při implemetaci využívat jednotného programovacího jazyka C#. Dalším důvodem je naše míra zkušeností s těmito technologiemi v porovnání s jinými. Výjimkou je použití SQL Serveru, který je pro nás novou zkušeností.

Pro implementaci webové aplikace využijeme technologii ASP.NET, konkrétně WebForms. Zamýšlené API realizujeme jako webovou službu, pro kterou použijeme technologii WCF. Data našich uživatelů pak budou uložena v databázi Microsoft SQL Server. Využití našeho API budeme demonstrovat na desktopové aplikaci vytvořené za použití Windows Forms. Následuje seznámení s jednotlivými technologiemi.

3.1.1 Všeobecně o technologii .NET

.NET Framework [2] je technologie společnosti Microsoft zastřešující celou platformu, která poskytuje nástroje a technologie pro vývoj, nasazení a spouštění desktopových, webových a mobilních aplikací nebo webových služeb. Jeho vznik se datuje do roku 2002 a dnes je dostupný již ve verzi 4.5. Jeho součástí je Common Language Runtime (CLR), který poskytuje například správu paměti a ostatní systémové služby. Dále pak rozsáhlá knihovna tříd, která představuje otestovaný znovupoužitelný kód pro všechny hlavní oblasti vývoje aplikací. Jejich funkce bude podrobněji popsána níže. Přehled architektury .NET Frameworku je zobrazen na obrázku 3.1 Přehled architektury .NET Frameworku.



Obrázek 3.1 Přehled architektury .NET Frameworku

CLR je komponenta .NET Frameworku, která je zodpovědná za kompilaci a spouštění .NET programů. Funguje jako vrstva mezi operačním systémem a aplikací napsanou v některém z .NET jazyků. Má na starosti správu vláken, správu paměti pro alokaci objektů a bufferů, Garbage Collection pro uvolňování nepotřebné paměti objektů a bufferů, dále pak typovou bezpečnost a správu výjimek. Jeho hlavní funkcí je kompilovat řízený kód do strojového a ten spustit. To se děje prostřednictvím dvou kroků. CLR nejdříve překompiluje zdrojový kód napsaný programátorem do procesorově nezávislého souboru instrukcí nazývaného Microsoft Itermediate Language (MSIL). Ten je pak možné efektivně, pomocí Just In Time (JIT) kompilátoru, kompilovat do strojového kódu.

Základní knihovna tříd také zvaná Base Class Library je rozsáhlá objektově-orientovaná knihovna tříd rozhraní a typů, které poskytují přístup k systémové funkcionalitě. Představuje tak základ, na kterém jsou založeny všechny .NET aplikace a komponenty. Tato knihovna funguje s kterýmkoli .NET programovacím jazykem. Všechny třídy jsou zde řazeny do logických celků tzv Namespace. Například pro práci s vstupně výstupními streamy nám stačí označit používání namespace System.IO.

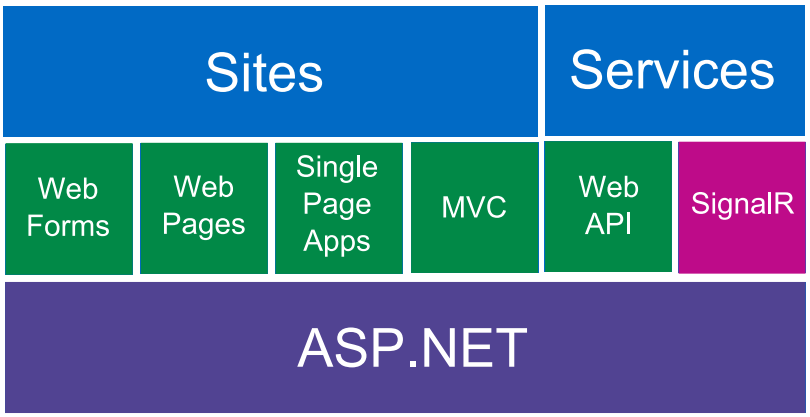
Následovat bude podrobnější seznámení s některými z použitých součástí technologie .NET Frameworku v našich aplikacích.

3.1.2 ASP .NET

ASP.NET je technologie na straně serveru pro vývoj webových aplikací a služeb. Vychází z klasického ASP, které byli součástí hned prvního .NET Frameworku, ale obě technologie jsou rozdílné. To v tom, že ASP.NET je postaven na CLR, takže programátor může použít jakýkoli .NET jazyk a na rozdíl od klasického ASP, kde je při každém požadavku stránka prováděna interpretem, je stránka v ASP.NET při prvním požadavku kompilována. Ostatní požadavky na stejnou stránku jsou již obsluhovány pomocí tohoto dll a tím dochází ke zrychlení oproti klasickým ASP. Další výhodou je, že kód je oddělen od designu a je tak přehlednější a jednodušeji rozšiřitelný. Na obrázku 3.2 je vidět více možností přístupu k tvorbě webových aplikací pomocí modelů. My v naši práci využíváme Web Forms.

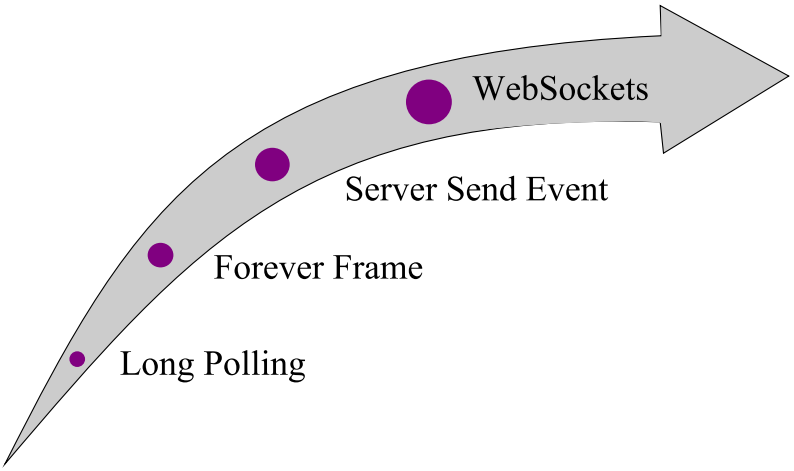
3.1.3 SignalR

Ve zkratce se jedná o novou knihovnu, která nám umožňuje jednoduše vytvořit real-time web pro .NET. SignalR začal jako víkendový projekt dvou vývojářů ASP.NET týmu v Microsoftu. Stojí za ním David Fowler a Damien Edwards. Projekt byl původně vyvíjen jako open source, po čase jej adoptoval Microsoft, kde už je pro něj vyčleněn vlastní tým, ale stále zůstal jako open source. Stal se tak oficiální součástí rodiny ASP.NET jak ukazuje obrázek 3.2 SignalR jako součást rodiny .NET.



Obrázek 3.2 SignalR jako součást rodiny ASP.NET

SignalR je jakousi abstrakci nad spojením klienta se serverem. Snaží se vytvořit spojení, které se nám jeví jako perzistentní. A tím mění pohled na dosavadní paradigma komunikace založeného na požadavku a odpovědi. Jeho hlavní možností je, že umožňuje serveru volaní klientských javascriptových metod. Dovoluje nám tak zprostředkovávat nejnovější informace uživateli bez jeho interakce. SignalR používá čtyři typy přenosů. Těmi jsou long polling, forever frame, server send events a WebSockets. Poslední dva jmenované jsou pak součástí nové specifikace HTML ve verzi 5.0, přičemž WebSockets je nový protocol označovaný jako ws:// nebo zabezpečená verze wss://. Vytvoření spojení klienta se serverem si pak SignalR sám domluví a vybere nejlepší možnou variantu přenosu. O veškerou režii se stará sám. Na obrázku 3.3 je vidět hierarchie priorit typů přenosů. V zásadě jde o to, že snahou SignalR je domluvit co možná nejlepší řešení, takže pokud nejsou k dispozici WebSockets snaží se vytvořit spojení pomocí server send events. To třeba nepodporuje prohlížeč Internet Explorer, takže v jeho případě se pokusí o spojení typu forever frame.



Obrázek 3.3 Hierarchie typů přenosů v SignalR

SignalR je tedy multi-platformní a multi-prohlížečové řešení real-time webu pro .NET s výhodou, že programátor může být osvobozen od programování na nižší úrovni. To je v SignalR realizováno tzv. Persistent Connection. Je zde, ale I možnost programování na vyšší úrovni abstrakce nad spojením v podobě Hubu.

V případě využití WebSockets se pak nejedná pouze o iluzi stálého spojení, ale je skutečně perzistentní. Pro využití WebSocket však potřebujeme na straně serveru mít podporu .NET frameworku 4.5 s IIS 8.0 a na straně prohlížeče[1] Firefox 6.0, Chrome 14.0 a Internet Explorer 10.0. SignalR pro použití s jinými typy přenosů je pak podporováno .NET frameworkem od verze 4.0. Na straně SignalR klienta může být:

- JavaScript
- .NET/WinRT
- Windows Phone
- Silverlight
- Neoficiální podpora je i pro Mono a iOS

SignalR je tedy podporován většinou internetových prohlížečů a nejběžnějšími typy platforem. Základní nastavení je však pro každého klienta individuální a liší se použitým typem spojení, od kterého se odvíjí jeho použitelnost a efektivita.

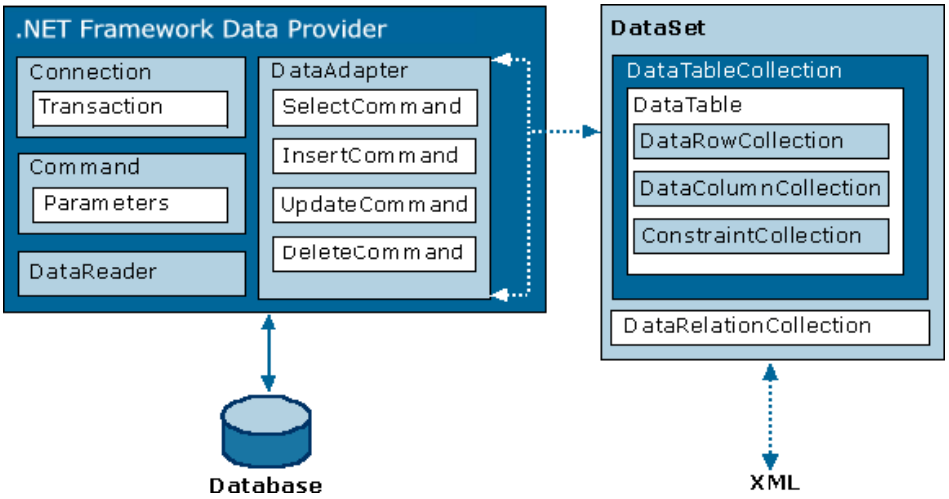
[1] Zdroj: http://caniuse.com/websockets

3.1.4 WCF

WCF je nástroj pro výtváření služeb a je součástí .NET Frameworku. Jedná se tedy o technologii využívající principů service-oriented architecture (SOA) k vytváření distribuovaných aplikací. Typicky toho můžeme využít v případě, pokud chceme sdílet naše data s jinými aplikacemi. Jestliže využívají naši službu nazýváme je klienti. Klienti mohou získávat data z více služeb, ale i služby mohou sdílet data mezi sebou. Klienti se připojují skrze tzv. endpoint. Ten má specifikovánu svou URL adresu, na které je dostupný a také vlastnosti definující jak budou data přenášena. Použitím WCF tedy můžeme posílat data jako asynchronní zprávy. Zprávy mohou být různých druhů od jednoduchých jako je samostatný znak nebo slovo v XML až po složité jako je proud binárních dat.

3.1.5 ADO.NET

ADO.NET znamená Microsoft ActiveX Data Objects a slouží pro komunikaci se zdroji dat jako může být například SQL Server (viz další kapitola) nebo také data zpřístupněná skrze OLE DB či dokonce XML. Aplikace používající tyto zdroje dat mohou ADO.NET použít pro připojení ke zdrojům a následně získávat, manipulovat a aktualizovat data.



Obrázek 3.4 Architektura ADO.NET [5]

Komponenty ADO.NET byly navrženy tak, aby oddělovaly přístup k datům a manipulaci s nimi. Přehled architektury si můžeme prohlédnout na obrázku 3.4 Architektura ADO.NET. Zde si můžeme všimnout zmíněných dvou hlavních komponent. DataSet pro přístup k datům nezávisle na jejich typu a druhou hlavní komponentou je Data Provider pro rychlou manipulaci s daty. Skládá se z dalších komponent jako je Connection, které se stará o připojení k databázi. Dále Command, který slouží například k získávání a úpravě dat nebo spouštění uložených procedur. DataReader poskytuje vysoce výkonný proud dat ze zdroje. A poslední součástí je DataAdapter, který slouží jako prostředník mezi zdrojem dat a DataSetem.

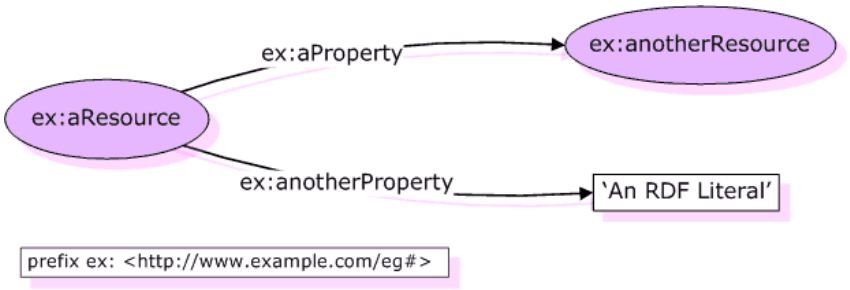
3.2 Sémantická data

Jak již bylo zmíněno výše chceme aby naše aplikace obsahovala funkcionalitu pro automatický návrh a podporu tvorby kvízových otázek s využitím sémantických dat. V této kapitole bychom rádi přiblížili základní pojmy této problematiky a take náš neúspěšný pokus o vlastní řešení.

3.2.1 Seznámení

S množstvím informací na internetu zároveň roste i obtížnost v jejich vyhledávání na základě klíčových slov. Přestože k tomuto účelu existují různé internetové vyhledávače, může být nalezení potřebných informací časově náročné a v některých případech i nemožné. Počítače totiž smyslům našich dotazů nerozumí. Zjednodušeně řečeno, většina vyhledávačů hledá podle nějakého typu podobnosti slov a frekvence jejich výskytu. Tuto skutečnost se snaží změnit sémantický web.

Sémantický web je rozšířením stávajícího internetu. Rozšířen je o to, že informace zde mají přesně definovaný význam, který umožňuje lepší komunikaci mezi uživatelem a počítačem. Celým základem sémantického webu je technologie RDF, která může být reprezentována pomocí XML. Zjednodušeně řečeno se jedná o kolekci trojic *(předmět, vlastnost, objekt)*. Tyto trojice popisují skutečnost, že *předmět* je *vlastností* nějakého *objektu*. Každá ze tří částí trojice může být popsána pomocí URI, což je jednoznačný identifikátor významu v tomto prostředí nebo v případě předmětu a objektu se může jednat o prázdný uzel a u objektu je i třetí možnost popisu doslovně nějakým řetězcem. Na následujícím obrázku 3.5 si můžete prohlédnout ukázku RDF trojic.



Obrázek 3.5 Ukázka RDF [6]

XML reprezentace obou trojic znázorněných na obrázku 3.5 pak bude vypadat následovně [6]:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://www.example.com/eg#">

  <rdf:Description rdf:about="http://www.example.com/eg#aResource">
    <ex:aProperty
rdf:resource="http://www.example.com/eg#anotherResource"/>
    <ex:anotherProperty>An RDF Literal</ex:anotherProperty>
  </rdf:Description>

</rdf:RDF>
```

RDF tedy představuje grafovou databázi dat, kde uzly jsou reprezentací *předmětů* a *objektů*, které jsou spojeny hranami představující *vlastnosti*. Nad tou můžeme vykonávat dotazy pomocí jazyka SPARQL. Tento jazyk se může zdát podobný jazyku SQL, protože se zde setkáme s klíčovými slovy jako je SELECT nebo WHERE, ale také s takovými, které z SQL neznáme jako OPTIONAL , FILTER a mnoho dalších. SPARQL dotaz se skládá z RDF trojic, kde předmět, vlastnost nebo objekt mohou být proměnné. Myšlenkou je porovnávat tyto dotazy s existujícími RDF trojicemi a najít tak řešení pro zadané proměnné. Dotazy se spouští nad nativní RDF databází nebo Relational Database to RDF (RDB2RDF) systému. Ty obyčejně poskytují endpoint pro spouštění dotazů a vrácení výsledku přes HTTP. Jako příklad takové databáze můžeme uvést DBpedii [7], která se zaměřuje na extrahování informací z Wikipedie.

3.2.2 Pokus o vlastní řešení

V naší webové aplikaci jsme plánovali vytvořit automatický systém pro návrh a podporu tvorby kvízových otázek s využitím sémantických dat. Za tímto účelem jsme provedli rozsáhlou rešerši. Jejim výsledkem bylo nalezení článku [3], který se nám velmi zamlouval. Na jeho základě jsme chtěli implementovat naše řešení. To se však jevilo jako náročný úkol. Proto po emailové konzultaci s Christinou Unger, jednou z autorů, jsme došli k závěru, že implementace celého řešení je příliš náročná. Autorka nám nabídla možnost, že konzultuje se svými kolegy napojení na jejich stávající system ovšem s tím, že by takové API museli nejprve vytvořit. My jsme se rozhodli, že pro naši práci se jedná spíše o okrajovou funkcionalitu ne její hlavní cíl. Proto i s ohledem na komplikovanost a časovou náročnost jsme tedy tuto variantu vyloučili a poohlédli se po jednodušším řešení.

Při dalším pátrání jsme narazili na článek [4], který oproti předchozímu vypadal, že by měl být mnohem jednodušší pro implementaci. Článek popisuje algoritmus pro dotazování DBpedie pomocí vět běžného jazyka. Pro definici příjímaných vět je zde prezentována gramatika (obrázek 3.6 Qedia – příjímaná gramatika). Ta příjímá jen velmi jednoduché věty jako například:

What animal that has the color that is gray eats leaves that belong to the species that is Eucalyptus?

```
Prop = Beg S P C
Beg = What | What does | What do
S = noun | S P.atr
C = noun | adjective | adverb | C P.atr
P.atr = that P C
P = verb
```

Obrázek 3.6 Qedia – přijímaná gramatika [4]

Ve větě nejdříve určíme slovní druhy (toto take není triviální úloha a součástí naší rešerše byl průzkum použití existujících řešení a jejich otestování). Následně se takto označenou větu pokusíme příjmout konečným automatem. Pokud ji automat přijme jeho výstupem by měl být RDF graf věty a následuje další faze algoritmu. V té máme vytvořené tři šablony SPARQL dotazů, kterými pak získáváme data z DBpedie. Za všechny uvedeme jeden příklad.

Věta: "What [property] has [subject]?"

Je přeložena do dotazu:

```
SELECT ?property WHERE
{
        :[subject] dbpedia:property ?property
}
```

Domnívali jsme se, že přijímání pouze jednoduchých vět je příklad toho, že popisovaný přístup získávání sémantických dat je z kategorie méně rozsáhlých a měl by tak splňovat naše požadavky na náročnost. Po počáteční snaze o implementaci konečného automatu pro přijímání výše zmíněné gramatiky jsme se rozhodli, že před dalším pokračováním důkladněji prostudujeme článek, ze kterého jsme vycházeli. Po jeho prostudování vyvstalo mnohem více otázek ohledně postupu v dalších fázích, nežli odpovědí. Domníváme se, že popsaný postup je popisován bez pro nás důležitých detailů. V druhé řadě jde také o původ článku, který podle našeho průzkumu nebyl zveřejněn na žádné konferenci, byl pouze publikován na internetu a není citován jinými autory. Při snaze o vyhledání dalších informací podle jeho jména jsme narazili na projekt, za kterým však stojí jiní autoři a navíc neodpovídá popisu zmiňovaného článku. Tento stín pochybnosti a množství času potřebného k realizaci s nejasným výsledkem nás donutilo ustoupit i od této varianty.

Dalším důvodem pro rezignaci na implementaci vlastního řešení vyhledávání v sémantických datech bylo zjištění ohledně jejich dostupnosti v rámci námi preferované oblasti databází. Ta bude hlavním tématem, pro kterou budeme chtít vytvářet kvízové otázky. Z našeho průzkumu vyplývá, že bohužel neexistuje žádná rozsáhlá databáze sémantických dat zaměřující se na tuto oblast. Proto by přínos tohoto řešení nebyl v požadovaném rozsahu.

3.2.3 Použité API

Po neúspěšných pokusech o vlastní implementaci jsme se rozhodli improvizovat a původně zamýšlenou funkcionalitu pro automatický návrh a podporu tvorby kvízových otázek s využitím sémantických dat v rámci možností zprostředkovat pomocí API třetích stran.

3.2.3.1 Bing Search API

Bing Search API je rozhraní umožňující aplikacím třetích stran využívat webového vyhledávacího nástroje jménem Bing společnosti Microsoft. Toto API je dostupné na Windows Azure MarketPlace. Kromě Search API nabízí Microsoft také API pro své mapy a překladač. Abychom jej mohli začít využívat musíme se registrovat. Po registraci si musíme zvolit typ licence jaký využijeme. Microsoft nabízí pouze jeden typ licence, který je zdarma. Ta nám umožňuje 5000 transakcí za měsíc. Pro naše účely by měla stačit.

Pro identifikaci aplikace, která API využívá zde existuje tzv. AppId. To je součástí query stringu pro každý pořadavek na API. V tomto query stringu můžeme nastavovat mnoho dalších parametrů od počtu výsledků až po jejich typ. Seznam typů je uveden v tabulce 3.1.

Typ	Popis
Web	výsledky hledání webových stránek
Images	hledání obrázků
News	výsledky pro zprávy
Videos	vyhledávání videí
Related Search	návrhy souvisejícího vyhledávání
Spelling Suggestions	návrhy při psaní na základě vloženého dotazu

Tabulka 3.1 Typy výsledků Bing Search API

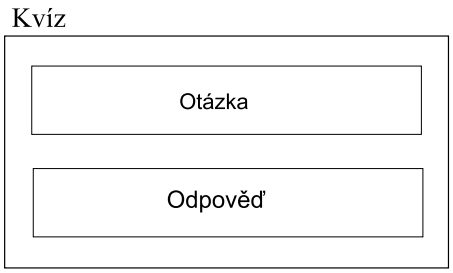
Microsoft poskytuje svůj vlastní wrapper pro práci s API, což činí jeho použití snadnějším. Výsledky jsou obdrženy ve formátu XML nebo Json.

3.2.3.2 WolframAlfa API

WolframAlfa je netypický vyhledávač, který nehledá stránky, na kterých by se mohla nacházet odpověď na vloženou otázku nebo vyhledávanou frázi, ale snaží se na otázku rovnou odpovědět. Využívá sémantických dat a také on nabízí programátorům své API. Jeho princip je podobný jako je tomu u Bingu. Po registraci získáme AppId, kterým se identifikuje naše aplikace. WolfrmaAlfa také nabízí wrapper pro své API, ale my jsme narazili na problémy při jeho stahování, takže jsme použili wrapper z jiného zdroje.

Zásadní rozdíl mezi API Bingu a WolframAlpha je ve formátu výsledku. V případě Wolfram Alfa jde o obrázky. Pouze v některých případech existuje možnost získat výsledek v jiném formátu. Těmi mohou být například plaintext nebo MathML.

3.3 Terminologie kvízů



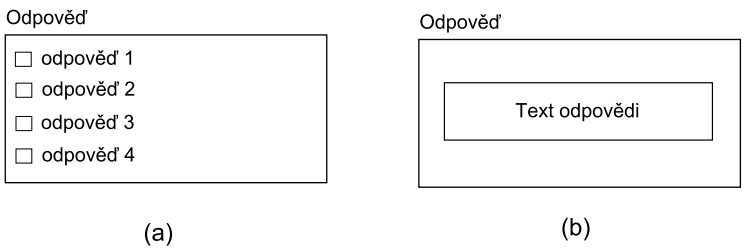
Obrázek 3.7 Schéma kvízové otázky

V tomto teoretickém úvodu bychom také chtěli uvést význam námi používané terminologie. Pokud v rámci analýzy a návrhu, později také implementace, mluvíme o kvízu nebo přesněji kvízových otázkách, máme na mysli základní jednotku, která zapouzřuje vše ostatní. Jak znázorňuje obrázek 3.7, kvíz obsahuje otázku a odpověď. Odpověď pak může být pěti druhů, které jsou popsány v tabulce 3.2.

Typ	Popis
select	odpověď se vybírá ze 4 možných
char	podpověď se vyplňuje do textového pole jako znaky (string)
number	podpověď se vyplňuje do textového pole pouze jako číslo (integer)
tuple	podpovědí je množina, kde jsou jednotlivé prvky odděleny čárkou
SQL	odpovědí je SQL dotaz

Tabulka 3.2 Druhy odpovědí.

Pro lepší představu může posloužit také obrázek 3.8. Na něm je zobrazena otázka typu select (a), tedy výběr z možností. U toho typu odpovědi může být více správných odpovědí. Všechny zbylé typy otázek si můžeme představit jako textové pole, do kterého vložíme odpověď. Na obrázku 3.8 znázorněno jako (b). Zde je jen rozdíl v pohledu na vložený text a to pro lepší možnosti vyhodnocení odpovědí.



Obrázek 3.8 Odpovědi typu (a) select a (b) ostatní

Kapitola 4

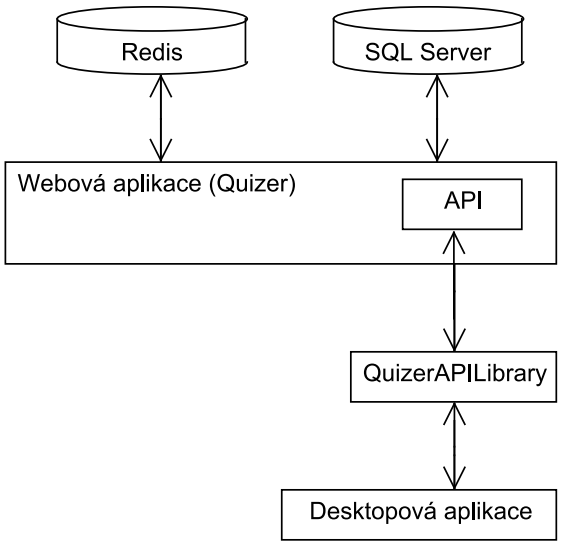
4. Popis aplikací

V této části práce se budeme zabývat popisem vytvářených aplikací. Ten je rozdělen do tří částí. V první části se budeme zabývat analýzou a návrhem, ve druhé pak vše popíšeme z pohledu implementace. Třetí část je zaměřena na nasazení webové apliakce. V kapitole 3.1 popisujeme základní technologie použité během vývoje.

4.1 Analýza a návrh

4.1.1 Architektura systému

Následující obrázek 4.1 popisuje schématické rozložení celého systému z pohledu rozložení implementovaných aplikací, knihoven, databáze a komunikace mezi nimi. Z obrázku je patrné, že API bude hostováno ve webové aplikaci.

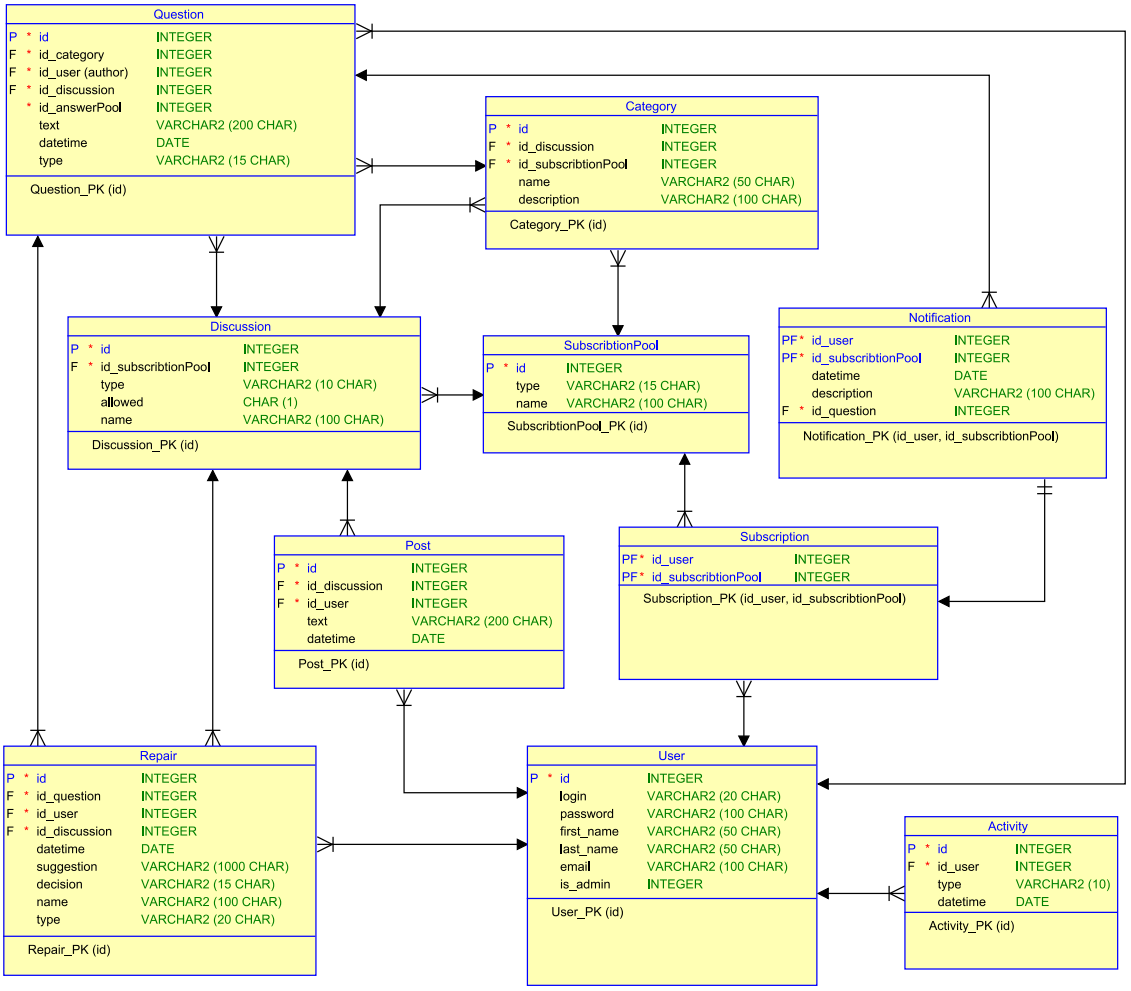


Obrázek 4.1 Architektura systému

4.1.2 Datová analýza

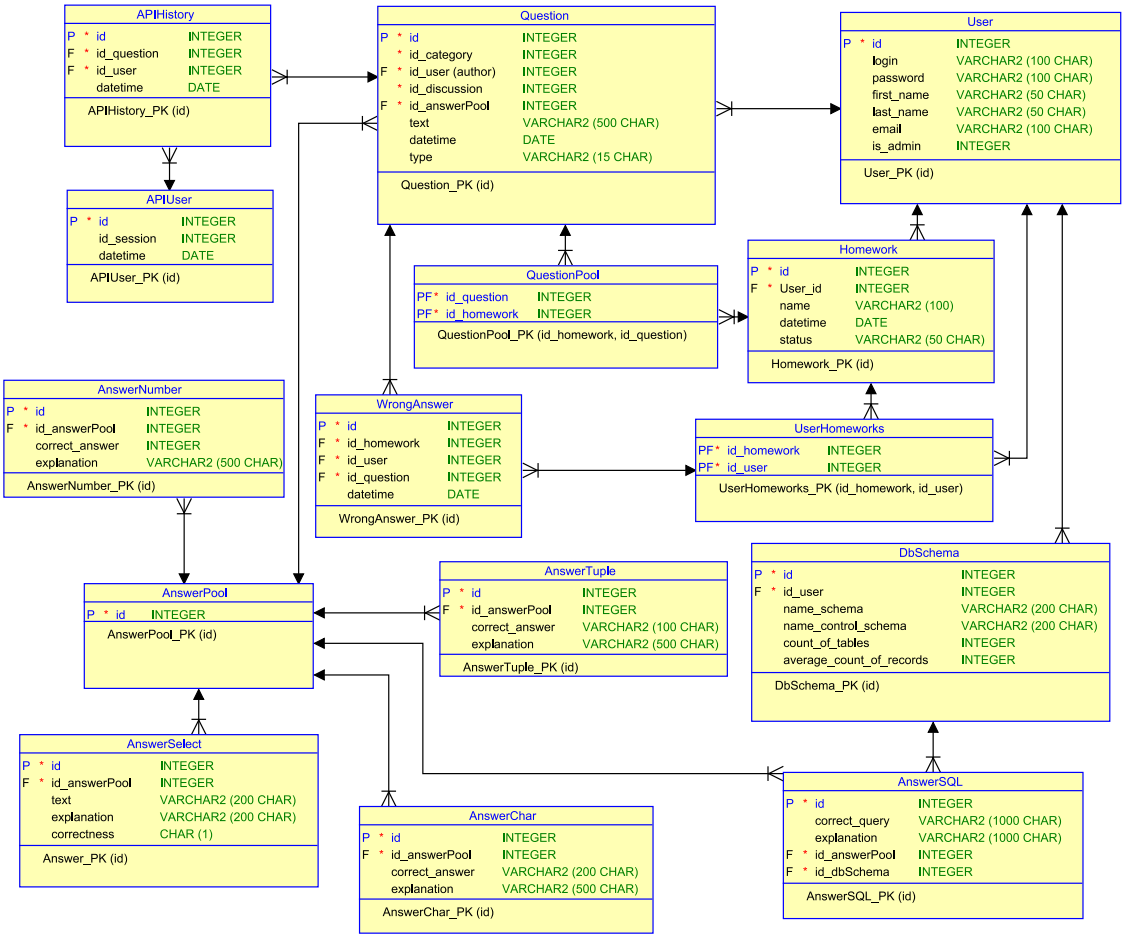
Důležitou součástí našeho řešení je databáze, ve které uchováváme všechna data. V našem případě je celkem rozsáhlá. To proto, že každý z hlavních funkčních požadavků vyžadoval uchovávání různých dat, pro které bylo potřeba vytvořit vlastní tabulky. ER diagram jsme pro přehlednost rozdělili do dvou částí. Ty jsou znázorněny na obrázcích 4.2a a 4.2b. ER diagramy na těchto obrázcích mají společné tabulky Question a User, které jsou ústředními prvky celého systému. Databázi kde budeme uchovávat většinu dat bude SQL Server. Pro data

kolem technologie SignalR bude využita databáze Redis označovaná také jako NoSQL databáze. Toto označení znamená, že se obyčejně jedná o jinou než relační databázi, která nevyužívá jazyk SQL. Redis je databáze založená na záznamech typu klíč - hodnota. Pod hodnotou si však můžeme představit i složitější datové typy než je string a to například List, Set nebo Hash. Další informace jsou dostupné zde [9].



Obrázek 4.2a ER-diagram databáze

Na obrázku 4.2a si můžeme prohlédnout ER diagram znázorňující schéma databázových tabulek pro funkcionalitu notifikačního systému (Notification, Subscription a SubscriptionPool), diskuzí (Discussion, Post), oprav (Repair) a kategorií (Category). V případě notifikačního systému má každá diskuze a kategorie svůj SubscriptionPool, pomocí kterého se uživatel přihlašuje k odběru novinek dané entity. Seznam těchto odběrů je v tabulce Subscription a jednotlivé upozorňovací záznamy pro notifikační systém jsou uloženy v tabulce Notification.



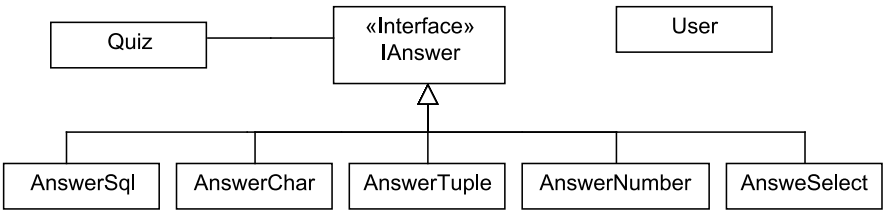
Obrázek 4.2b ER-diagram databáze

ER diagram na obrázku 4.2b znázorňuje tabulky pro uchovávání dat všech druhů odpovědí (AnswerNumber, AnswerChar, AnswerTuple, AnswerSelect a AnswerSQL spolu s DbSchema). Tyto odpovědi jsou dostupné přes tabulku AnswerPool. Důvod použití nové tabulky namísto cizího klíče na tabulku Question ve všech tabulkách odpovědí je takový, že tímto řešením umožníme případnou budoucí možnost sdílení odpovědí mezi kvízovými otázkami. Další funkcinalitou pokrytou tímto diagramem s pohledu databázové analýzy je funkce API (APIUser, APIHistory) a vytváření a testování domácích úkolů (QuestionPool, Homework, UserHomeworks a WrongAnswer). Domácí úkol je uložen v základní tabulce Homework. Přiřazení studenti pak v tabulce UserHomeworks. Tabulka QuestionPool slouží pro uložení seznamu otázek domácího úkolu a tabulka WrongAnswer je využívána při vyhodnocení špatné odpovědi na otázku pro přehled její obtížnosti.

Pro tuto databázi vytvoříme objektově relační mapování. Jeho součástí bude i podpora transakcí. ORM bude vytvořeno jako samostatná knihovna. To proto, aby byla zajištěna lepší použitelnost v případě potřeby jejího využití další aplikací.

4.1.3 Knihovna LibraryOfQuizer

Tato knihovna obsahuje třídy sjednocující práci s objekty systému sdílené mezi jednotlivými aplikacemi nebo komponentami. Její obsah si můžeme prohlédnout na obrázku 4.3, na kterém je znázorněn třídní diagram. Jeho smyslem je znázornit jednotnou práci s více typy odpovědí skrze rozhraní IAnswer.



Obrázek 4.3 LibraryOfQuizer – třídní diagram

4.1.4 Webová aplikace

Hlavní aplikací našeho řešení je webová aplikace. Ta bude umožňovat veškerou práci s daty a komunikaci mezi uživateli. Seznam hlavních funkčních požadavků můžete vidět v tabulce 4.1.

Kód požadavku	Popis
WEB_001	Aplikace umožní registraci a přihlašování uživatelů.
WEB_002	Uživatelé budou moci vytvářet kvízy.
WEB_003	Uživatelé budou mít množnost navrhovat opravy kvízů.
WEB_004	Uživatelé budou moci mezi sebou komunikovat v diskuzích.
WEB_005	Kvízy budou řazeny v kategoriích.
WEB_006	Správce bude mít možnost upravovat hierarchii kategorií.
WEB_007	Aplikace bude umožňovat vytváření domácích úkolů.
WEB_008	Uživatelé budou moci plnit své domácí úkoly.
WEB_009	Aplikace bude mít systém na zasílání, zobrazování a kontrolu upozornění na aktivity týkající se uživatele v reálném čase.
WEB_010	Uživatel bude mít možnost vybírat si kategorie, ve kterých chce být upozorňován na novinky.
WEB_011	Aplikace umožní vyhodnocení odpovědi na kvíz.
WEB_012	Aplikace bude umožňovat vytvoření vlastního schématu v databázi a spouštění SQL dotazů nad ním pro kvíz typu SQL
WEB_013	Komunikace v diskuzích bude probíhat v reálném čase bez znovu načtení stránky.

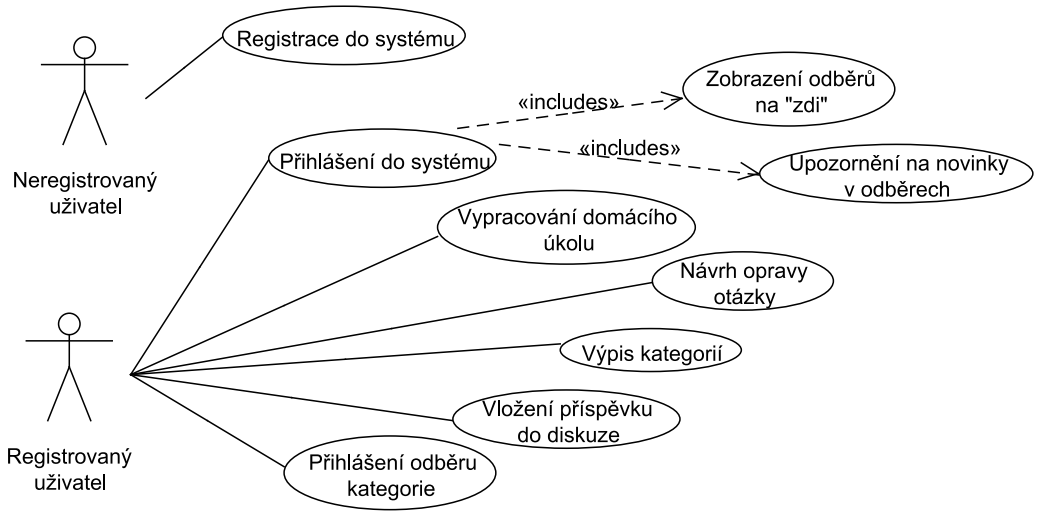
Tabulka 4.1. Seznam hlavních funkčních požadavků na webovou aplikaci.

V našem systému budeme mít celkem 4 role, jejichž seznam je znázorněn v tabulce 4.2. Role Autor je v systému používána ve všech případech, ve kterých je uživatel autorem. Těmito případy rozumíme pokud je uživatel autorem kvízu, domácího úkolu nebo navržené opravy. Ve všech těchto případech je v daném kontextu uživatel v roli autora.

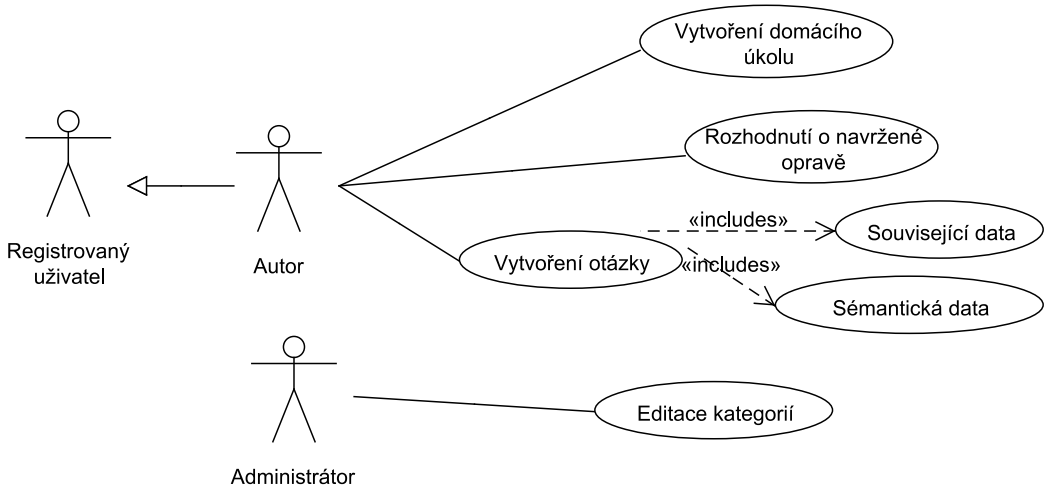
Role	Popis
Neregistrovaný uživatel	Role pro neregistrované uživatele.
Registrovaný uživatel	Základní role pro uživatele.
Autor	Role pro situace, ve kterých je uživatel zároveň autorem.
Administrátor	Role pro autorizaci k editaci kategorií.

Tabulka 4.2. Role v systému

Na obrázcích 4.4 a 4.5 si můžeme prohlédnout dostupnou hlavní funkcionalitu pro každou z rolí popsanou use case diagramy.



Obrázek 4.4 Registrovaný a neregistrovaný uživatel – Use case diagram

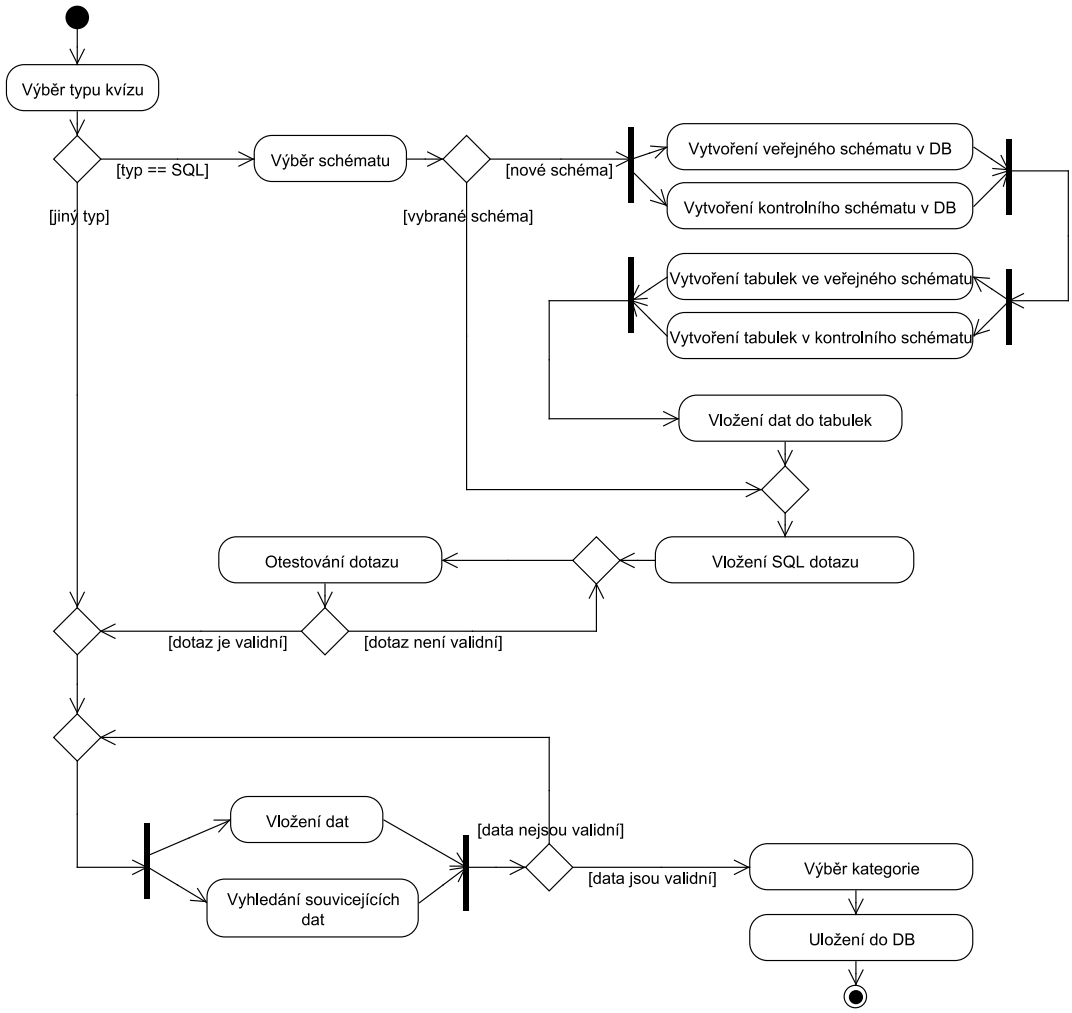


Obrázek 4.5 Autor a administrátor – Use case diagram

V následujících podkapitolách popíšeme některé netriviální funkce z pohledu analýzy a návrhu.

4.1.4.1 Vytvoření kvízové otázky

V tomto případě se jedná o jednu z nejdůležitějších funkcí systému. Ta bude umožňovat vytvářet kvízové otázky, které jsou základem celé sociální sítě. Jak již bylo zmíněno v kapitole 3.4. aplikace umožňuje vytvářet kvízové otázky 5 různých druhů. Postup vytváření kvízu je uveden na obrázku 4.6 Vytvoření kvízu – aktivitní diagram. K tomto diagramu je potřeba zmínit, že proces *Vyhledání souvisejících informací* zahrnuje procesy *Související data* a *Sémantická data* z obrázku 4.5, které budou ve webové aplikaci reprezentovány funkcemi pro využití API Bingu a API WolframAlfa. Jak již bylo zmíněno dříve v případě použití těchto API se jedná o náhradu za původně zamýšlené vlastní řešení ve vyhledávání v sémantických datech. Další funkcí zahrnutou v procesu *Vyhledání souvisejících informací* pak bude generování podobných SQL dotazů. Tato funkcionalita bude podrobněji popsána v kapitole 4.1.4.5 SQL Parser.

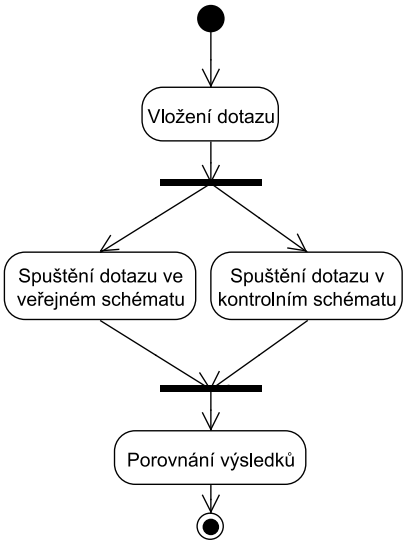


Obrázek 4.6 Vytvoření kvízové otázky – aktivitní diagram

Největší režii tedy potřebuje vytvoření kvízu typu SQL. Jedním z funkčních požadavků uvedených v tabulce 4.1 je požadavek WEB_012 na to, aby se tyto dotazy nevyhodnocovali pouze na porovnání dvou řetězců, ale aby bylo uživateli umožněno dotazy testovat. Proto musíme autorovi takového kvízu umožnit vytvořit tabulky se svými daty. Zde vyvstává otázka ohledně bezpečnosti. Abychom zajistili bezpečnost naší databáze izolujeme uživateli vytvářené tabulky a vkládaná data v samostatných schématech. Dále bude také zapotřebí vytvořit pro uživatele databázovou roli s náležitými právy.

Dalším důležitým faktorem je řízení připojení k databázi a vykonávání SQL dotazů prostřednicvím aktuální role uživatele v systému. Musíme zajistit, aby uživatelé nemohli provádět změny v jiných schématech. Pro tyto účely vytvoříme více druhů připojení. Pokud je uživatel autor otázky bude se připojovat svým unikátním connection stringem. V jiném případě se k vykonání dotazů bude využívat základní connection string s právy pouze pro čtení.

Pro zajištění větší kvality kontroly uživateli vládaných dotazů budeme poskytovat následující funkcionalitu. Jelikož bude mít jakýkoli uživatel možnost spouštět SQL dotazy nad daným schématem a systém mu vrátí výsledek pro jeho dotaz, mohlo by dojít k situaci, že uživatel využije jiných dotazů k tomu, aby poskládal SQL dotaz tak, že bude vracet správný výsledek, ale nebude proveden podle zadání. Tomu se budeme snažit zabránit tím, že během tvorby kvízu bude vytvořeno kontrolní schéma, do kterého má autor možnost vložit jiná data. Toto kontrolní schéma je obyčejnému uživateli skryto. Vyhodnocení dotazu pak budeme kontrolovat v obou schématech zároveň jak znázorňuje obrázek 4.7. Toto řešení nám dovoluje předejít dříve popsaným podvodům.



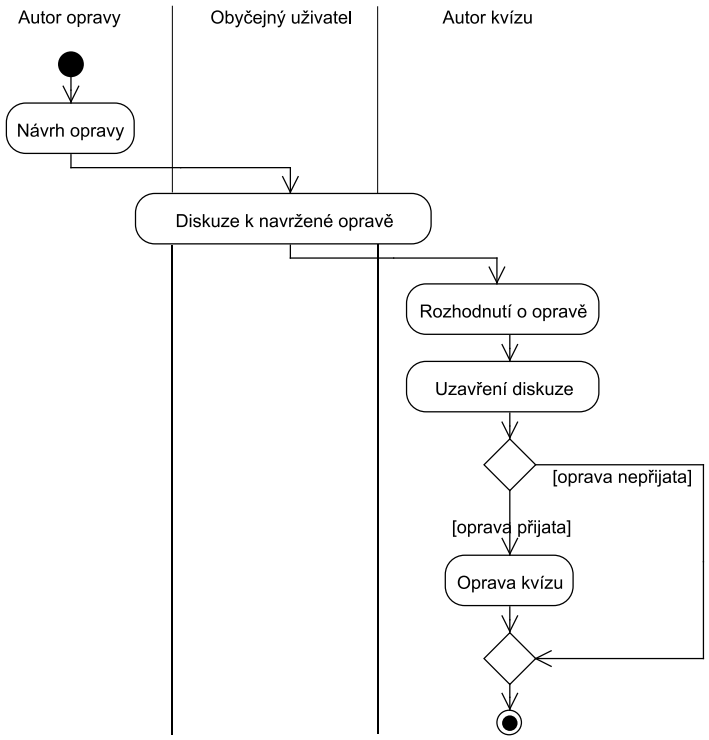
Obrázek 4.7 Konrola správnosti dotazu – aktivitní diagram

4.1.4.2 Systém nápověd pro domácí úkoly

Naši aplikaci bychom chtěli reálně využít jako učební nástroj pro procvičovaní znalostí studentů. Pro zvětšení efektivity učebního procesu studentů jsme se na základě vědecké publikace [1] rozhodli implementovat část tohoto konceptu. Tato část je založena na otázce se sadou odpovědí s možností výběru jedné z nich. Za normálních okolností by student odpověď označil a poslal k vyhodnocení domácího úkolu, čímž by vše skončilo. V tomto případě však vyhodnocení testu probíhá tak, že pokud student odpověděl špatně, pro jeho zvolenou odpověď se zobrazí vysvětlení, které je pro každou odpověď unikátní. Toto by mělo vysvětlit v čem udělal chybu a následně je mu umožněno znovu zopakovat domácí úkol. Při každé další špatné odpovědi (za předpokladu, že vybral jinou odpověď než v předchozím případě) získává stále vice nápověd. Pro jednu správnou odpověď se doporučuje vytvořit tři špatné. To je však možné pouze u otázek typu výběr z možností. V našem řešení se vyskytuje celkem 5 typů otázek a proto jsme původní myšlenku na ostatních typech otázek museli mírně modifikovat. Zde tak student dostane pro každou svou odpověď pouze jednu nápovědu. Popsaný postup by tak měl klást důraz na učení a procvičování nepochopených témat namísto pouhého otestování znalostí.

4.1.4.3 Návrh opravy a její posouzení

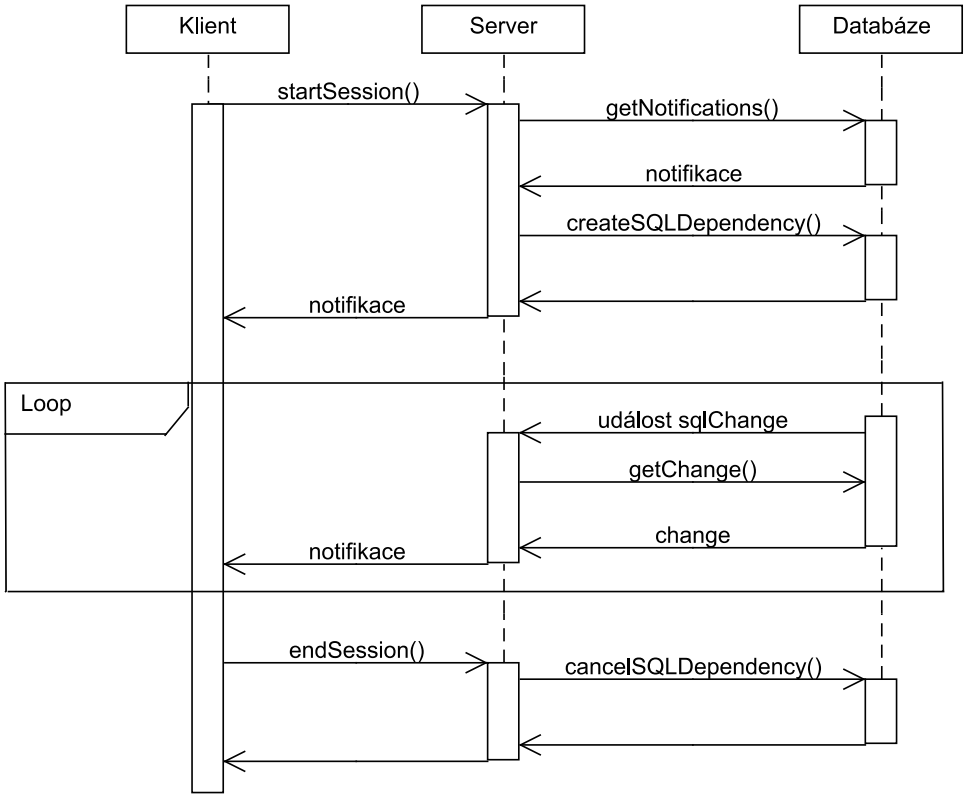
Funkčním požadavkem WEB_003 z tabulky 4.1, je aby uživatelé sociální sítě měli možnost jednotlivým kvízovým otázkám navrhovat opravy. Oprava může být netriviálního charakteru a tím také komplikovaná pro vysvětlení v jednom odstavci textu s různými odkazy na další zdroje. Z tohoto důvodu budeme ke každé opravě poskytovat možnost diskuze. Diskutovat o opravě budou moci všichni uživatelé. Stránky oprav tak musí být všem dostupné. Jejich seznam bude dostupný z detailu jednotlivých kvízových otázek. Posouzení těchto návrhů pak bude na autorovi otázky. Ten bude moci opravu buď schválit a uskutečnit nebo opravu zamítnout. Po rozhodnutí o opravě bude diskuze uzavřena. Celý životní cyklus opravy znázorňuje následující obrázek 4.8.



Obrázek 4.8 Aktivitní diagram návrhu a posouzení opravy

4.1.4.4 Notifikační systém

Jako jednu z požadovaných vlastností systému (WEB_009 v tabulce 4.1) máme vytvořit systém pro zasílání zpráv s novinkami uživateli. Tyto zprávy mají upozorňovat na aktivitu, která se nějakým způsobem týká uživatele. Může se jednat o novou zprávu v diskuzi, které se účastní, novou opravu kvízové otázky jejíž je autorem anebo také nový kvíz ve sledované kategorii. Ve všech těchto případech chceme na tuto aktivitu uživatele upozornit. Jak už vyplývá ze sekvenčního diagramu na obrázku 4.9 pro náš systém je zapotřebí nalézt řešení, které hlídá změny v databázi a umožňuje zasílat zprávy ve směru ze serveru na klienta a to bez přímého vytvoření požadavku klienta.



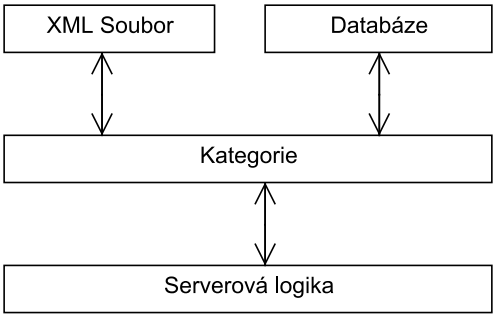
Obrázek 4.9 Notifikační systém – sekvenční diagram

Za tímto účelem použijeme technologii SignalR ve spolupráci s SqlDependency, což je součást technologie ADO.NET. Po přihlášení uživatele se nejprve zjistí všechny změny, na které jej chceme upozornit po dobu jeho nepřítomnosti v systému. Následně se přihlásíme ke sledování změn výsledků dotazu v databázi pomocí SqlDependency. Pokaždé pokud k takovéto změně dojde ihned pomocí SignalR pošleme upozornění uživateli. Po jeho odhlášení zrušíme sledování změn pro daného uživatele. Pokud k manuálnímu odhlášení nedojde zruší se odběr novinek na základě vypršení časového inervalu funkčnosti (tzv. timeout) SqlDependency automaticky.

4.1.4.5 Hierarchie kategorií

Funkční požadavky (WEB_005 a WEB_006 v tabulce 4.1) pro tuto funkcionalitu požadují, aby bylo možné spravovat kategorie v hierarchii. Proto je s tímto v analýze třeba počítat. Pro vytvoření hierarchie nám přijde optimální využít XML datový soubor, který ji bude udržovat. V XML souboru bude hierarchie tvořena zanořováním jednotlivých uzlů. Každá kategorie tedy bude představovat uzel, který bude identifikován pomocí id a jména. Jediné tyto dva atributy budou replikovány jak v XML souboru tak i v databázi. V tomto případě tedy budeme řešit i problematiku uložení informací na více místech, dat v databázi a hierarchii v XML souboru. Proto bude potřeba vytvořit prostředníka, který zajistí komunikaci s oběma úložišti.

Dalším důvodem využití XML souboru pro hierarchii kategorií je použití komponenty třetí strany jménem ASTreeView [18]. Ta zajišťuje systém drag and drop a také grafickou reprezentaci dat. Datovým vstupem této komponenty je právě XML soubor.

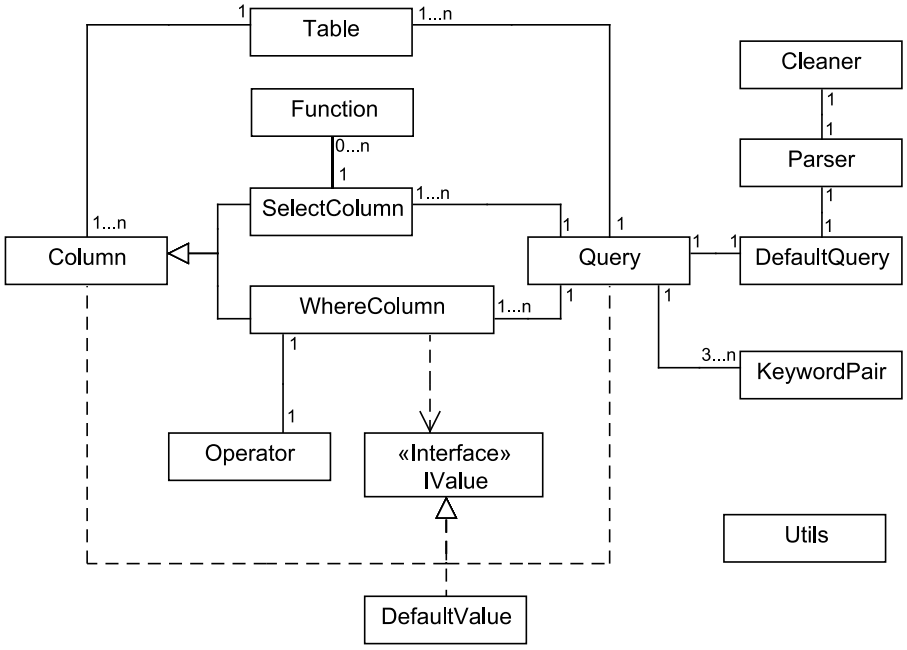


Obrázek 4.10 Schéma kategorií

Při provádění změn v kategoriích musíme řešit také možnost paralelního souběhu mezi více uživateli. To buď uzamykáním XML souboru a povolením editace pouze jednomu uživateli nebo celou funkcionalitu vytvoříme jako tzv. kolaborační dokument pomocí technologie SignalR. Obě varianty mají své klady i zápory. V případě, kdy zvolíme variantu uzamykání editace pouze pro jednoho uživatele, je hlavní výhodou to, že nedojde k žádné chybě způsobené neaktuálními daty jako je například špinavé čtení nebo ztráta aktualizace. Nevýhodou však je možnost editace pouze jedním uživatelem v tomtéž čase. Tuto nevýhodu můžeme vyřešit kolaboračním dokumentem, kde budou moci uživatelé zároveň paralelně upravovat kategorie. U této varianty se však vystavujeme riziku, že pokud uživatelé zároveň upravují stejný uzel XML dokumentu, může dojít ke ztrátě aktualizace. V našem řešení jsme se rozhodli, i podle očekávané četnosti editace hierarchie, použít variantu uzamykání pouze pro jednoho uživatele.

4.1.4.6 SQL Parser

V rámci vyhledávání souvisejících informací během vytváření kvízových otázek vytvoříme také funkcionalitu pro generování podobných SQL dotazů. Tuto funkcionalitu budeme realizovat pomocí SQL Parseru. V implementaci se zaměříme na 5 druhů změn, jmenovitě syntaktické změny, negace klíčových slov a operátorů , změny spojení, změny hodnot a záměny agregačních funkcí . Přínosem tohoto řešení je hlavně použití zmiňovaného schématu. Schéma je kontextem dotazu. Takže v případě generování jiných hodnot a změn atributů spojení budeme moci vytvářet záměny za reálné hodnoty a atributy, což by mohlo přinést uživateli zjednodušení vytváření kvízových otázek. Použití bude probíhat tak, že uživatel nejprve vloží svůj vlastní originální dotaz, vybere k němu související schéma a vybere druh požadované změny a nechá je vygenerovat. Z popisu vyplývá, že podmínkou pro funkčnost je tedy existence vlastního schématu. Hlavním účelem SQL Parseru tedy bude převést dotaz z textového do objektového formátu, ze kterého následně budeme moci generovat požadované změny. Třídní diagram SQL Parseru je znázorněn na obrázku 4.11.



Obrázek 4.11 Třídní diagram pro SQL Parser

```
S = AFI | AFNRT
A = select E
B = C column D
C = table. | alias. | ε
D = ,B | ε
E = W(B) | B
F = from G
G = table alias H | table as alias H
H = ,G | ε
I = join table alias on J | join table as alias on J
J = M column L M column K
K = and J | or J | ε
L = < | > | <> | = | <= | >= | X like
M = table. | alias.
N = where O
O = C column L C value Q | C column X between value and value Q |
    C column P (S)Q
P = X in
Q = and O | or O | ε
R = having O | ε
T = order by C column UV | ε
U = ,C column U | ε
V = desc | asc | ε
W = count|sum|avg|first|last|max|min|ucase|upper|lcase|lower|len
X = not | ε
```

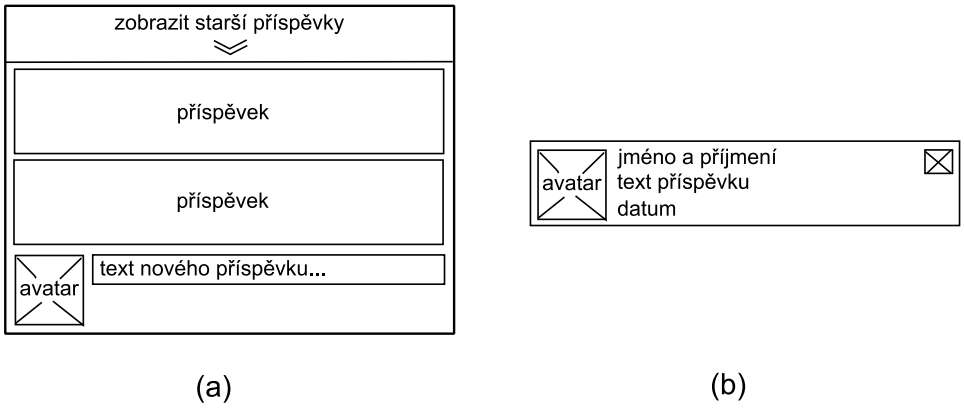
Obrázek 4.12 SQL Parser - příjímaná gramatika

Vzhledem ke komplikovanosti různých možných podob SQL dotazů není naší snahou je pokrýt všechny. Omezíme se pouze na některé ze základních tvarů a zápisů SQL dotazů. Druh podporovaných dotazů je popsán přijímající gramatikou na obrázku 4.12. Velké písmena označují neterminály. Slova psaná malými písmeny a znaky pak označují terminály.

4.1.4.7 Návrh komponent Discussion a Post

Jedním z funkčních požadavků (WEB_004) popsaných v tabulce 4.1 je uživatelům umožnit komunikaci v diskuzích. Tento požadavek má za důsledek to, že v celé webové aplikaci bude na různých místech diskuze. Pokud bychom měli poskládat požadované serverové komponenty a potřebný kód na každém takovém místě, vedlo by to k jejich nepřiměřené replikaci. To nás vede k vytvoření vlastních komponent pro tyto účely. Výhody jsou jednoznačné. Replikace kódu je omezena na minimum a případné změny stačí provést na jednom místě a projeví se všude, kde jsou komponenty používány.

Na obrázku 4.13 je znázorněn návrh komponent. Komponenta Discussion se skládá z příspěvků, které jsou reprezentovány vlastní komponentou Post. Dále je zde textbox pro vkládání nových příspěvků. To se bude provádět na základě stisknutí tlačítka enter. Komponenta Discussion pak bude umožňovat schovávání starších příspěvků, aby existovala možnost zpřehlednit stránku. Diskuze i příspěvky jejichž je uživatel autorem se mu budou přizpůsobovat. To znamená, že avatar neboli obrázek uživatele bude aktuální pro toho daného uživatele a u jeho příspěvků bude v pravém horním rohu možnost jejich smazání.



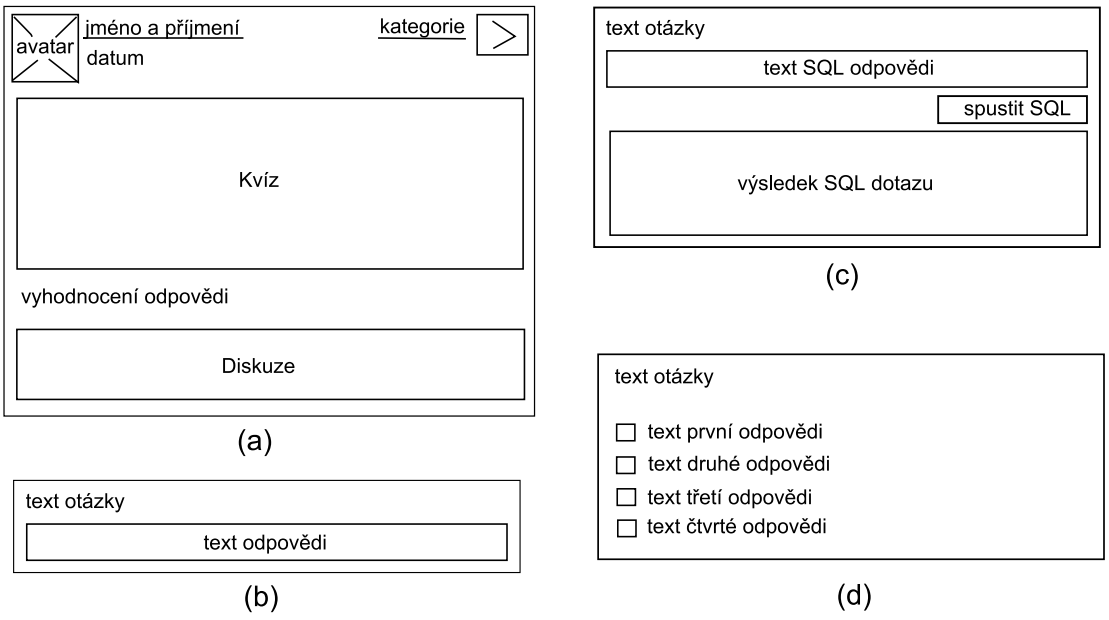
Obrázek 4.13 Návrh komponent (a) Discussion a (b) Post

4.1.4.8 Návrh komponent Stream a Quiz

Jelikož hlavním stavebním kamenem obsahu našeho systému je kvíz, bude potřeba vytvořit komponentu pro jeho zobrazování. Důvody jsou stejné jako je tomu v předchozí kapitole a navíc se tím zjednoduší práce s nimi. Pro naše účely musíme uvažovat dvojí možné použití kvízu. První je zobrazení jeho detailu, ve vyhledávání nebo na "zdi". Druhé jeho použití pak je v rámci vykonávání domácího úkolu, kde se stane součástí testu. V tomto případě se informace o kvízu zobrazované uživateli omezí pouze na otázku a odpověď. Kdežto u prvního případu bude množství informací mnohem větší.

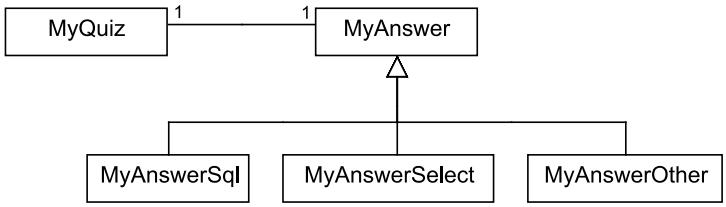
Komponenty vytvoříme tedy dvě. Tu hlavní pro všechny informace budeme dále nazývat Stream a druhou varinatu Quiz. Komponenty jsou navrženy tak, aby se komponenta Quiz mohla stát součástí komponenty Stream. Ta bude kromě jiných informací také obsahovat další naši komponentu Discussion. Zmíněnými dalšími informacemi myslíme vlevo nahoře avatara uživatele, který je autorem kvízu. Ve formě odkazů na detaily budou reprezentovány autorovo jméno a příjmení a také kategorie kvízu, do které je zařazen. Vpravo nahoře bude tlačítko zpřístupňující menu s dalšími možnostmi.

Pro komponentu Quiz bude potřeba vytvořit tři její varianty. Dáno je to rozdílem v typech kvízů, které bude představovat. Pro typy char, number, tuple bude použita varianta (a) z obrázku 4.14, pro druh sql varianta (c) a pro typ select zbylá varianta (d). V případě varianty (c) pro kvíz typu SQL bude před vyhodnocením odpovědi dostupná možnost spustit odpověď ve formě SQL dotazu. Jeho výsledek bude formátovaný do tabulky a bude se zobrazovat v do té doby neviditelném poli na obrázku označovaném jako výsledek SQL dotazu.



Obrázek 4.14 Návrh komponent (a)Stream a (b,c,d)Quiz

Komponenta kvízu bude implementována podle třídního diagramu na obrázku 4.15. Základem je třída MyQuiz, která bude pro jednotnou práci se všemi typy využívat abstraktní třídy MyAnswer. Tu budou rozšiřovat již zmíněné tři typy odpovědí kvízu.



Obrázek 4.15 Komponenta Quiz – třídní diagram

4.1.5 API

API diplomové práce slouží jako komunikační prostředek mezi naší databází a aplikacemi třetích stran. API budeme realizovat jako webovou službu za pomocí technologie WCF viz kapitola 3.1.4. Základním požadavkem na API je, aby bylo možné získávat data jednotlivých kvízů, tedy texty otázek i jejich odpovědí. Pro získané kvízy bude API umožňovat i vyhodnocení jejich zodpovězení. Seznam hlavních funkčních požadavků je uveden v tabulce 4.3.

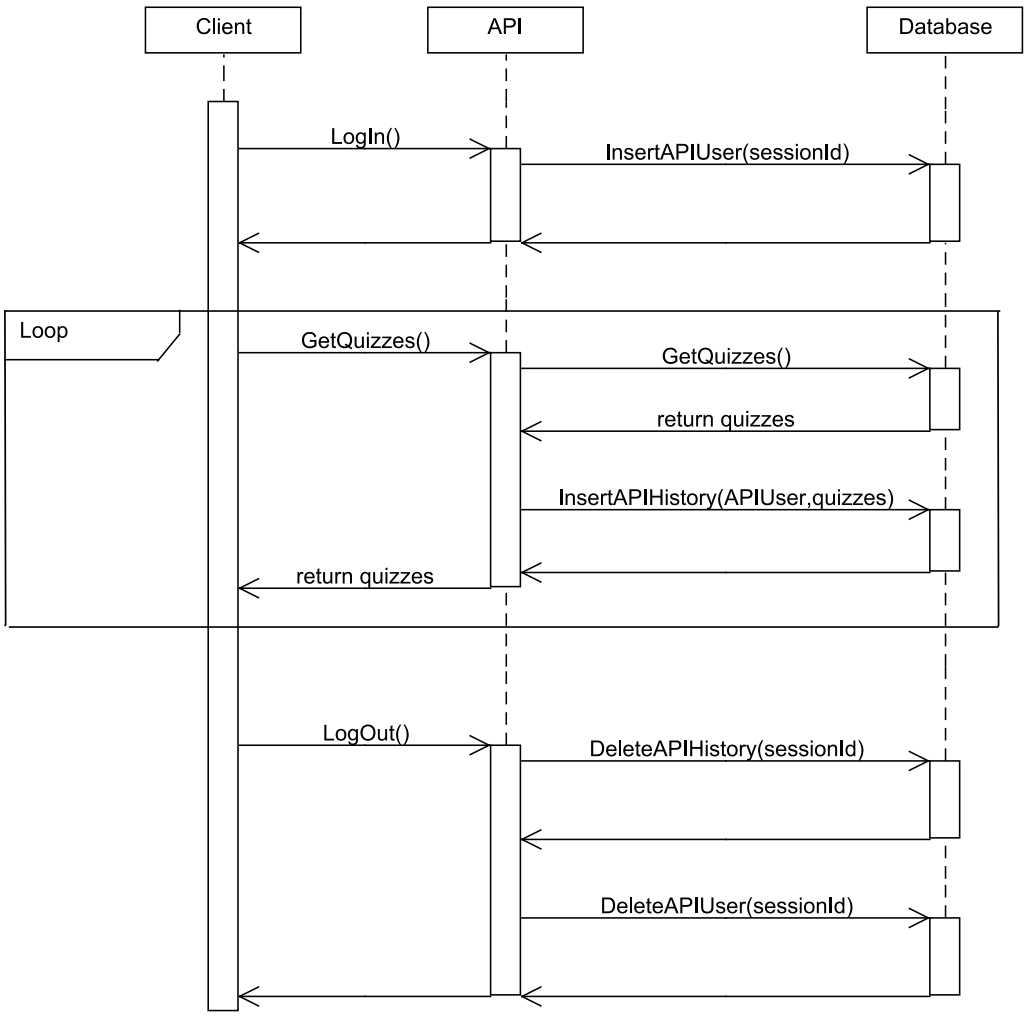
Kód požadavku	Popis
API_001	API bude umožňovat klientovi získávat náhodně vybrané kvízy z databáze.
API_002	API bude schopno vyhodnotit správnost odpovědí pro jednotlivé kvízy zaslené klientovi.
API_003	API umožní identifikaci připojeného klienta.
API_004	API bude mít kontrolu nad již poslanými kvízy klientovi.
API_005	Odesílané kvízy budou ve formátu XML.

Tabulka 4.3 Seznam hlavních funkčních požadavků pro API

V první fázi analýzy a návrhu našeho API jsme počítali s nejjednodušší variantou, která by aplikacím třetích stran umožňovala pouze získávat kvízové otázky a následně vyhodnocovat jejich odpovědi. Postupem času však vznikla spolupráce s jednou z bakalářských prací. Její motivací je vytvořit hru založenou na kvízech s možností sázek na své odpovědi. Vyskytla se tedy možnost reálného využití našich dat touto aplikací.

Pro splnění korektnosti pravidel soutěžní aplikace bylo důležité, aby nedošlo k situaci, kdy během jednoho soutěžního kola, kvízy získané prostřednictvím API obsahovaly některý kvíz více než jednou. Pokud bychom pouze modifikovali SQL dotazy mohlo by dojít k tomu, že volání metody k získání kvízů pro soutěžní kolo více než jednou, by opět mohlo mít za následek přítomnost stejných kvízů. Proto, abychom za každé možné situace mohli garantovat splnění tohoto požadavku, bylo potřeba přehodnotit původní návrh API.

První změnou bylo vytvořit WCF službu s použitím session, abychom umožnili identifikaci unikátního klienta. V dalším kroku bylo potřeba vytvořit nové tabulky v databázi, které budou uchovávat data o přihlášených klientech a jím poslaných kvízech. Tyto tabulky se jmenují APIUser a APIHistory viz. obrázek 4.2b. Následně jsme mohli upravit SQL dotazy tak, aby vybírali pouze takové kvízy, které již nejsou obsaženy v APIHistory. Průběh komunikace pak zobrazuje obrázek 4.16 Sekvenční diagram pro API.



Obrázek 4.16 Sekvenční diagram pro API

4.1.6 Desktopová aplikace

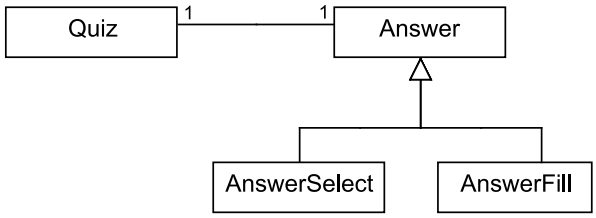
Součástí našeho řešení je také vytvoření desktopové aplikace pro demonstraci práce s poskytovaným API. Seznam hlavních funčních požadavků je uveden v tabulce níže.

Kód požadavku	Popis
DES_001	Aplikace umožní přihlášení a odhlášení k API.
DES_002	Aplikace umožní získat data prostřednictvím API.
DES_003	Uživatel bude moci odpovídat na získané kvízy.
DES_004	Aplikace bude schopna vyhodnotit správnost zadaných odpovědí.
DES_005	Aplikace bude obsahovat knihovnu pro práci s API.
DES_006	Aplikace bude obsahovat komponenty pokrývající všechny druhy kvízů.

Tabulka 4.4. Hlavní funkční požadavky na desktopovou aplikaci

4.1.6.1 Komponenta Quiz

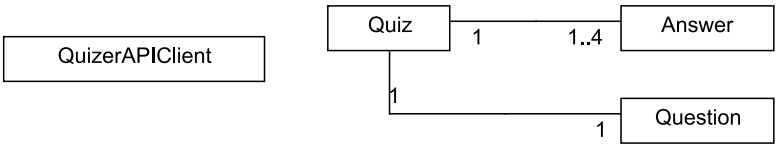
V případě komponenty desktopové aplikace pro zobrazování kvízu budeme na rozdíl od komponenty webové aplikace potřebovat pouze dva typy kvízů. To je dáno možnostmi API, které v případě kvízu typu sql neposkytuje možnost spouštění vlastních dotazů a tím má pro konzumenta API stejné parametry jako kvízové otázky typu char, number a tuple. Všechny zmíněné typy jsou sjednoceny do komponenty AnswerFill. Třídní diagram pro tuto komponentu je zobrazen na obrázku 4.17. Důvodem proč API neposkytuje možnost spouštění vlastních dotazů je množství aplikací a jejich uživatelů, kteří využívají API. Jejich identifikace se provádí pouze pomocí session, která není navázána na žádného uživatele a má tedy jenom jednu definici connection stringu pro připojení k databázi. V případě spouštění vlastních dotazů přes API by u tohoto spojení bylo potřeba pro každý dotaz měnit základní schéma, nad kterým se spouští. To by znamenalo zásadní snížení propustnosti systému, které je nežádoucí. V naší webové aplikaci bude tento problém vyřešen tak, že každý uživatel má pro spouštění dotazů vygenerován vlastní connection string.



Obrázek 4.17 Komponenta desktop Quiz - třídní diagram

4.1.6.2 Knihovna QuizerAPILibrary

V případě distribuce desktopové aplikace pro ukázku použití API bude její součástí knihovna ulehčující práci s webovou službou. Knihovna bude tak jako ukázková aplikace dostupná pouze pro platformu .NET. Třídní diagram této knihovny je zobrazen na obrázku 4.18.



Obrázek 4.18 Knihovna QuizerAPI – třídní diagram

Třída QuizerAPIClient slouží pro komunikaci se službou a práci s XML. Výsledkem je extrahování informací pro každý kvíz do přehledné třídy Quiz.

4.2 Implementace

4.2.1 Použité nástroje, knihovny a komponenty třetích stran

V této kapitole vyjmenujeme všechny nástroje, knihovny a komponenty třetích stran, které jsme v naší práci použili.

- Microsoft Visual Studio 2012
  je nejnovější verze vývojového prostředí Microsoftu. My budeme k implementaci používat jazyk C#.

- Microsoft SQL Server 2008 R2
  původně jsme zamýšleli použít verzi 2012, ale vyskytl se problém se stahováním z MSDNAA tak jsme využili této varianty.

- Microsoft Server 2008 R2 Datacenter
  je operační systém serveru, který jsme použili na virtuálním stroji našeho produkčního prostředí.

- MS Open Tech port Redisu 2.4
  původně unixová slovníková databáze. My jsme však použili její port pro Windows.

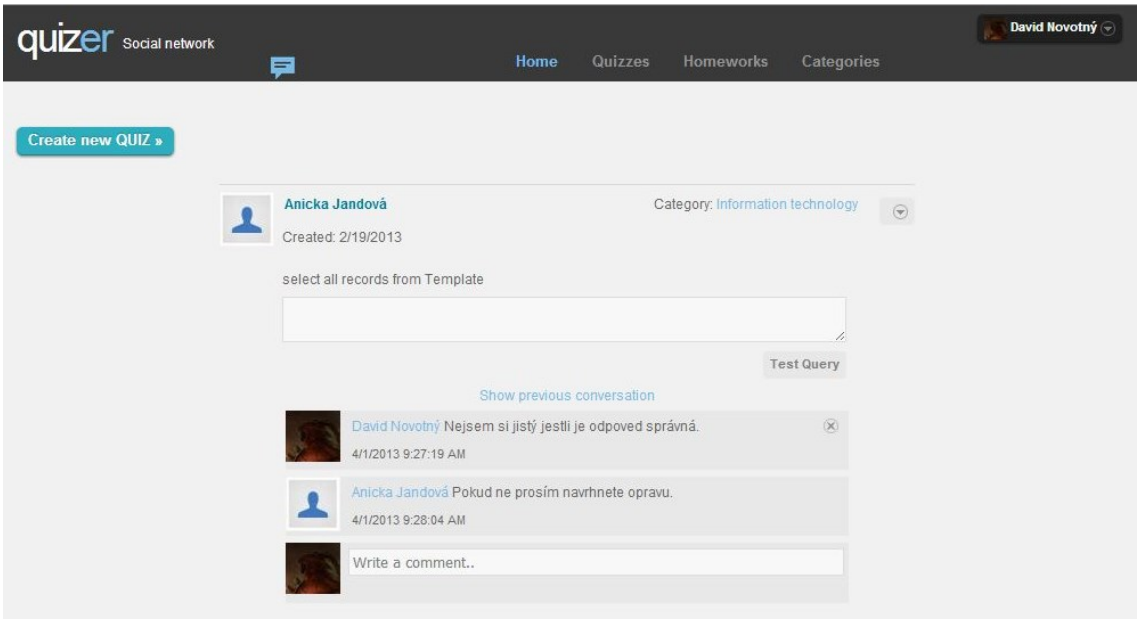
- ServiceStack.Redis
  je opensource C#.NET klient pro Redis.

- ASTreeView
  je komponenta [18] pro práci s XML umožňující vyjádřit jej ve stromové struktuře. Umožňuje jeho editaci uživatelsky příjemným systémem Drag And Drop.

- Gimp 2.8.4
  je program pro práci s obrázky.

4.2.2 Vzhled webové aplikace

Na obrázku 4.19 je možné si prohlédnout vzhled webové aplikace. Konkrétně se jedná o stránku hlavního streamu. Na obrázku je možné všimnout si také komponenty Stream společně s komponentou Discussion.

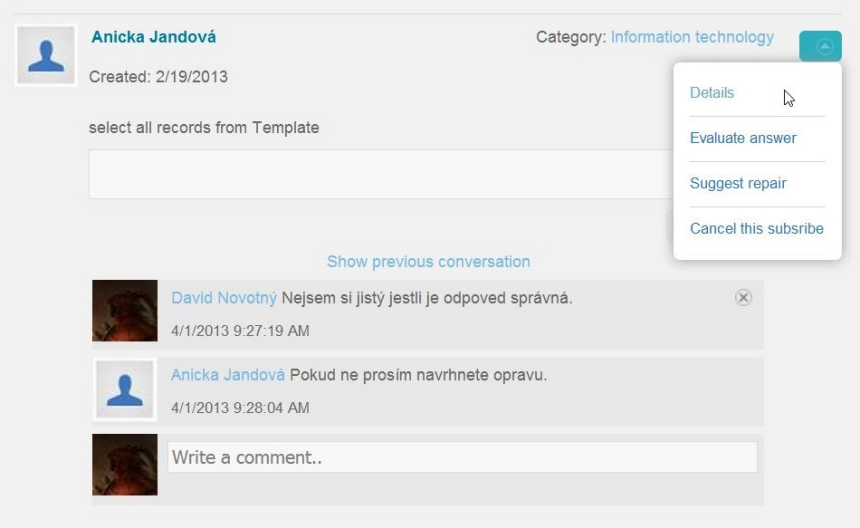


Obrázek 4.19 Vzhled webové aplikace

Během implementace jsme kladli velký důraz na prezentační část. Snažili jsme se o jednotný vzhled a líbivé barevné téma. Pro jeho dosažení jsme použili CSS, javascript a jQuery. V některých případech jsme použili nebo rozšířili řešení třetích stran, ať už se jedná o CSS úpravu tlačítka [13] či jiných [14], [15], [16]. Pro některé části aplikace, jako jsou obrázky například notifikace (blíže později), jsme použili pro jejich tvorbu program Gimp.

4.2.3 Vlastní komponenty obecně

Vlastní komponenty jsou naprogramovány pomocí Server Control. Jelikož obsah a některá funkčnost komponent je závislá na tom, jaký uživatel je bude využívat, používají knihovnu tříd, kterou sdílí s webovou aplikací. V této knihovně nazývané LibraryOfQuizer jsou obsaženy třídy pro práci s daty a hlavně jejich formátování. V závislosti na komponentě jsou instance těchto tříd součástí konstruktorů našich komponent. Na jejich základě pak nastavujeme různé vlastnosti komponent jako například autentizaci pro dostupnost některých funkcí, získání obrázku avatara uživatele nebo URL jeho profilu. Komponenty se skládají z dalších elementárních komponent dostupných v ASP.NET. Celá komponenta je pak pomocí HtmlTextWriteru vyrenderována zároveň se styly přesně podle jejich návrhu v kapitolách 4.1.4.7 a 4.1.4.8. Jejich formátování bylo obtížné a testování vzhledu časově náročné. Každá změna si totiž vyžádala kompilaci a znovu spuštění stránky. Díky tomu, že jsou celkem komplikované a tak je jejich formátování v kódu náročné a náchylné na chyby. Náročné je to tím, že formátovací chyby projdou debugerem a my si jich všimneme až na výsledné vykreslené stránce v prohlížeči. Ukázka komponenty Stream s otevřeným kontextovým menu je zobrazena na obrázku 4.20.



Obrázek 4.20 Komponenta Stream

Při implementaci vlastních komponent jsme je testovali v separátní stránce, kde jsme pro jednoduchost a důraz na komponentu nepoužili MasterPage ani žádné styly. Díky tomuto postupu nastala chyba při předělání do reálné stránky již s MasterPage, s různými kontejnery a styly. Na této stránce najednou přestaly fungovat javascripty. Po bližším zkoumání jsme přišli na to, že ASP.NET používá systém pro pojmenování id komponent takový, který se v základním nastavení chová následovně. Představme si situaci kdy na stránce máme kontejner pro obsah, který bude vložen na MasterPage. V tomto kontejneru máme Panel, do kterého chceme programově přidat například Label. V kódu všem komponentám nastavíme unikátní id, aby nedocházelo ke kolizím a v javasriptu napíšeme kód, ve kterém pomocí těchto přiřazených id chceme manipulovat s komponentou. Zmíněné základní nastavení se postará o to, že výsledné HTML zaslané klientovi bude obsahovat jiná id než jsme komponentám přiřadili my. Vypadat bude tak, že bude rekurzivně složeno z id všech kontejnerů, ve kterých se nachází, oddělených podtržítkem. V případě zmiňovaného Labelu id bude vypadat takto: idContent_idPanel_idLabel. Tomuto chování se dá zabránit nastavením vlastnosti ClientIDMode komponent na Static. Tím zachováme generování id v takové formě v jaké jsme je sami zapsali. Tímto jsme tedy umožnili javasriptu hledat komponenty na stránce podle id námi definovaného.

Na další problém jsme narazili při testování událostí komponent vložených z CodeBehind. Tehdy nastala situace, kdy se při vyvolaném PostBacku stránky nefungovalo zachycení události na vložené komponentě. Po delším zkoumání byl důvod zřejmý. Když jsme programově z CodeBehind přidali komponentu poprvé, měly určité vygenerované id. Při PostBacku se v životním cyklu stránky musí komponenty znovu programově přidat na stránku. Tady nastal problém v tom, že tentokrát měly komponenty jiné id a tak jsme se snažili zachytit událost na komponentě, která už neexistovala. Řešením bylo ručně nastavit všem koponentám unikátní id. Tím po rekontrukci stránky při PostBacku existovaly komponenty stejně pojmenované jako před ním a nedošlo tak ke ztracení události na komponentě.

4.2.4 Komponenty Discussion a Post

V první fázi implementace jsme nejdříve použili obyčejný Server Control. Do něj jsme chtěli referencovat knihovnu AjaxControlToolkit, ale narazili jsme na problémy s její registrací. Tyto problémy vyřešilo předělání celé komponenty do Ajax Server Control. Využití Ajaxu spočívalo v tom, že jsme staré příspěvky schovávali pomocí komponenty Expander a celá diskuze byla obsažena v komponentě UpdatePanel. Ten jsme chtěli využít proto, aby uživatel nemusel aktualizovat celou stránku pro získání nejnovějších příspěvků v diskuzi.

Jednou z komplikací se ukázala být registrace metod add_expandComplete a add_collapseComplete v javascriptu pro každý Ajax Expander. Původně jsme měli pouze jeden, na který jsme se odkazovali z javascriptu přímo. To se ukázalo jako nevhodné řešení v případě, že těchto komponent bude na stránce více. Následně jsme zjistili, že pro každý CollapsiblePanelExtender musíme registrovat tyto metody s jejich vyhledáním namísto document.getElementById(), který hledá pouze v DOM elementech musíme použít $find(), která představuje Sys.Application.findComponent a vztahuje se na jakoukoli komponentu knihovny Microsoft AJAX Library, která byla přidána programově. Na základě fóra[2] jsme tuto situaci vyřešili programovým vytvořením javascriptové metody onLoad znázorněné ve výpisu 1. Důležité také bylo uvědomit si, že tyto AJAX Extendery v javascriptu nehledáme podle jejich id, ale behaviorID.

```
private void registerBehaviorForDiscussion()
{
       ArrayList ar = new ArrayList();
       FindControlsRecursiveByType(ar, Page, typeof(MyStreamQuiz));
       string scriptstr = "function pageLoad(sender,args){";
       foreach (object control in ar)
       {
              string behaviorID =
              ((MyStreamQuiz)control).discussionControl.collapsePanel.BehaviorID;

              scriptstr += "$find('" + behaviorID +
              "').add_expandComplete(expandHandler);";
              scriptstr += "$find('" + behaviorID +
              "').add_collapseComplete(collapseHandler);";
       }
       scriptstr += "}";
       ScriptManager.RegisterStartupScript(Page, this.GetType(), "script",
       scriptstr, true);
}
```

Výpis 1 Registrace událostí CollapsiblePanelExtenderu

Základní chybou tohoto návrhu bylo, že jsme počítali pouze s variantou kdy na celé stránce je pouze jedna diskuze. Další problémy nastaly v případě, kdy se na stránce zobrazilo

[2] zdroj dostupný na http://forums.asp.net/p/1294064/2507458.aspx

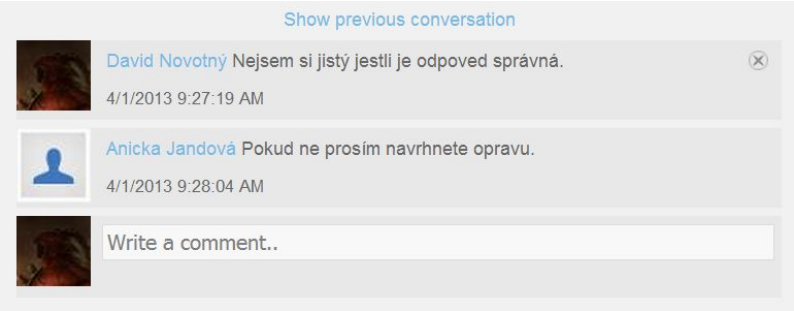
více diskuzí, z kterých každá měla svou vlastní režii updatePanelu a jejich aktualizace měla velký vliv na odezvu stránek. Řešením tohoto problému bylo využít technologii jQuery a SignalR, které společně pro uživatele zajišťují úplně stejnou funkcionalitu. Díky tomu však bylo zapotřebí komponenty kompletně přepsat. Bylo potřeba začít od začátku a na místo AjaxServerControl použít obyčejný ServerControl. Zachytávání událostí pro odeslání příspěvku po stisknutí tlačítka enter se přesunulo z codeBehind do javascriptu. Bylo nutné vytvořit nový SignalR Hub, abychom napsali kód zajišťující podobnou funkcionalitu pro jakou jsme používali UpdatePanel.

Tato implementace má mnohem menší režii a díky tomu i lepší odezvu stránek, takže na uživatele nepůsobí negativně jako tomu bylo v předchozím případě. Hrubou chybou při implementaci komponent pro diskuzi bylo neuvědomění si jejich možný celkový počet na stránce. Díky tomu jsme kompletně naimplementovali chybné řešení. Implementace diskuze se tak protáhla minimálně na dvojnásobek a my jsme tím ztratili drahocenný čas.

Jelikož je vložení příspěvku spouštěno javasciptem, bylo nutné zohlednit a zabezpečit jeho možné zneužití. Toho jsme dosáhli tak, že v Hubu na straně serveru před jakýmikoli akcemi je nejprve prováděna kontrolní autentifikace uživatele. To provedeme díky contextu, který je dostupný v každé metodě Hubu. Ten obsahuje property Context.User.Identity.Name, která je nastavena díky použití FormsAuthentication při přihlašování uživatele.

Pro stížení manipulace s javasriptem případným narušitelům jsme přesunuli formátování HTML příspěvků na stranu serveru. Po odeslání zprávy tak hub nerozesílá jenom text zprávy autora a čas, ale celý příspěvek naformátovaný v HTML.

Pokud nastane situace, kdy se uživatel nemůže připojit k hubu využíváme naše staré řešení, na které je při odchycení této chyby při startu stránky uživatel přesměrován. Při požadavku na další stránku se opět pokoušíme o spojení s hubem. Výsledný vzhled komponent si můžeme prohlédnout na obrázku 4.21.



Obrázek 4.21 Vzhled komponent Discussion a Post

4.2.5 Komponenta Quiz s odpovědí typu SQL

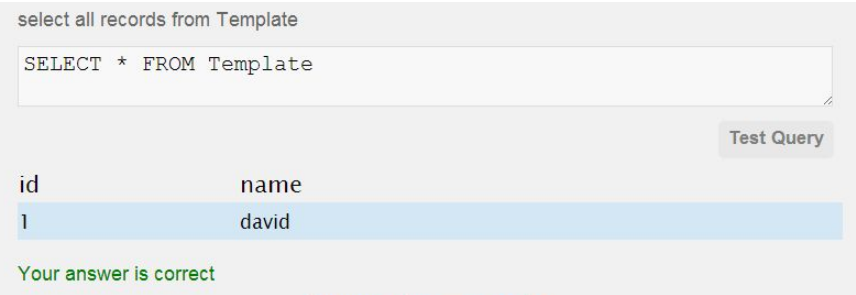
Pro spouštění vlastních dotazů pro odpovědi typu SQL bylo potřeba každému uživateli hned při registraci do systému vytvořit vlastní login i v databázi. To proto, abychom jej mohli připojovat přes jeho unikátní connectionString. Tato potřeba vznikla na základě toho, že ke spuštění dotazu musíme uživateli změnit defaultSchema na to, které je aktuální pro daný kontext kvízu. Pokud bychom toto řešili přes základní připojení, docházelo by ke kolizím mezi spouštěnými dotazy a různými schématy.

Jak již bylo popsáno v analýze, schémata jsou důležitá pro izolování dat uživatelů. Dalším významným aspektem jsou práva, která bude mít uživatel dostupná. Jelikož uživatel může spouštět jakýkoli dotaz, který napíše, museli jsme pro bezpečnost vytvořit také databázovou roli, kterou přiřazujeme hned při vytváření jeho loginu. Část nastavovaných práv je vidět ve výpisu 2.

```
CREATE ROLE my_enable_create_schema AUTHORIZATION QuizerDatabaseUser
GRANT CREATE SCHEMA TO my_enable_create_schema;
GRANT SELECT TO my_enable_create_schema;
GRANT EXECUTE ON setDefaultSchemaToUser TO my_enable_create_schema
WITH GRANT OPTION
GRANT EXECUTE ON setDboSchema TO my_enable_create_schema WITH GRANT
OPTION
GRANT EXECUTE ON CompareResults TO my_enable_create_schema WITH GRANT
OPTION
DENY SELECT ON SCHEMA::[dbo]TO my_enable_create_schema;
```

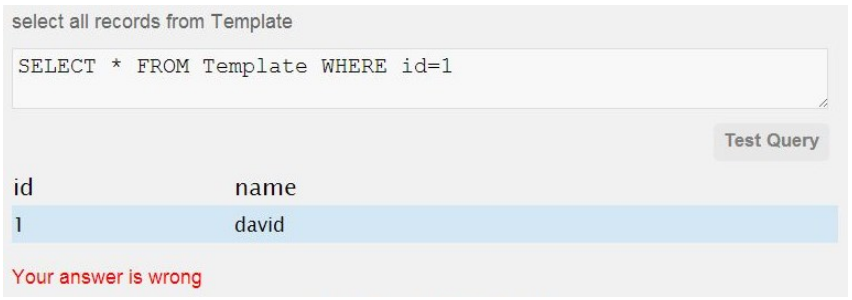
Výpis 2 Nastavení práv role pro uživatele

Součástí funkcionality poskytované u kvízové otázky typu SQL je kontrola správnosti podle kontrolního schématu pro prevenci zneužití SQL dotazů. Toto budeme demonstrovat na jednoduchém příkladu zobrazeném na obrázku 4.22. Tam si můžeme všimnout správné odpovědi na otázku, která požaduje výběr všech řádků z tabulky Template.



Obrázek 4.22 Kvíz typu sql - správná odpověď

Tímto dotazem jsme však zjistili jaké záznamy jsou v tabulce a v případě nějakého složitějšího kvízu bychom mohli podobným postupem sestavit dotaz, který vrací stejný výsledek, ale nesplňuje zadání. Tuto situaci v našem jednoduchém příkladu zobrazuje obrázek 4.23. Dotaz vrací stejný výsledek pro schéma dostupné uživateli, ale díky rozdílným výsledkům v kontrolním schématu je odpověď vyhodnocena jako špatná.



Obrázek 4.23. Kvíz typu sql - špatná odpověď

4.2.6 ORM a změna databáze

Na začátku implementačního procesu, kdy jsme ještě neměli připravené produkční prostředí pro nasazení našeho systému, jsme používali školní Oracle databázi na adrese dbsys.cs.vsb.cz. Důvodem tohoto výběru byla předešlá zkušenost s databázemi Oracle a její rychlá dostupnost bez potřeby instalace. V této databázi jsme vytvořili kompletní databázové schéma s testovacími daty. Dále jsme naprogramovali první verzi objektově relačního mapovaní pro naši webovou aplikaci skrze providera Oraclu.

Problém přišel až po vytvoření produkčního prostředí, kde jsme se rozhodli nasadit databázi Microsoft SQL Server. To z důvodu jednodušší administrace při instalaci a její menší celkové velikosti. Přepsat původní ORM nebyl takový problém. Pouze si to vyžádalo další čas navíc. Problém přišel až s přepisováním triggerů, které jsme používali v databázi Oracle. Tam jsme používali triggery typu BEFORE INSERT jehož alternativa se v T-SQL používaného v SQL Serveru nenachází. Nejprve jsme stejné funkcionality docílili použitím triggeru typu INSTEAD OF INSERT. Toho jsme však dosáhli jinými postupy než v Oraclu. Další problém, ale nastal v případě, že jsme potřebovali id právě vloženého záznamu. Id totiž generuje sama databáze při použití klíčového slova IDENTITY. Tento systém generování ID je méně náročný než v případě oraclu, protože tam jsme to řešili sekvencemi a dalšími triggery. Nevýhodou však je, že při použití INSTEAD OF triggeru v něm k tomuto id přístup nemáme. Proto bylo nutné pro jejich získání v ORM provádět dotazy podle právě vkládaných údajů, což mohlo mít nejednoznačný výsledek v případě stejných hodnot více záznamů. Z tohoto důvodu následoval další přepis. Namísto použitých triggerů tentokrát do transakcí v ORM. Tady jsme využili možnosti, že součástí vkládaného dotazu může být vrácení id aktuálně vloženého

záznamu. Ukázka je znázorněna ve výpisu 3. Timto se snížil počet dotazů typu select volaných nad databází.

```
//Deklarace dotazu
String SQL_INSERT = "sql insert dotaz" + " SET @newId=SCOPE_IDENTITY()";


//Přidání parametru
command.Parameters.Add("@newId",SqlDbType.Int).Direction =
ParameterDirection.Output;


//Provedení příkazu a získání id
command.Prepare();
command.ExecuteNonQuery();
if (command.Parameters.Contains("@newId") == true)
{
        newId = Convert.ToInt32(command.Parameters["@newId"].Value.ToString());
}

return newId;
```

Výpis 3 Získání id vkládaného záznamu

Objektově relační mapování je naprogramováno s využitím parametrizovaných dotazů a podporou transakcí. ORM nejprve využívalo jenom webová aplikace a v ní hostovaná webová služba API. Nyní je však implementováno jako samostatná Class Library. To proto, že jej využívají i naše komponenty skrze další Class Library, která sdružuje třídy pro práci s daty, ktere využívá jak webová aplikace tak i naše komponenty.

4.2.7 Autorizace úpravy obsahu

Editaci obsahu pouze oprávněným uživatelům zajišťujeme tak, že jsme pro všechny možnosti editace vytvořili zvláštní stránku, která je volně dostupná. Na této stránce pak podle hodnoty id, která je součástí URL a je dostupná přes kolekci QueryString, získáme hodnotu id pro prvek, který chceme upravovat. Prvkem v tomto kontextu rozumějme obsah webu vytvořený uživatelem, který jako jediný má právo jej jakkoli upravovat nebo mazat. Takovým obsahem může být kvíz, navržená oprava a domácí úkol nebo v případě role admin úprava kategorií. Díky tomuto řešení vzniká bezpečnostní hrozba ve formě úpravy QueryStringu uživatelem, který by se tak bez kliknutí na potřebný odkaz na stránce mohl dostat k úpravě cizího obsahu. Abychom uživatelům v tomto zneužití zabránili využíváme následujícího postupu.

U každého prvku, kterého je uživatel autorem, vkládáme menu pro jeho správu. Kontrolu nad vyrenderováním takového menu máme my na straně serveru, takže můžeme snadno zajistit abychom jej programově vygenerovali pouze oprávnenému uživateli. Menu se skládá z tlačítek, u kterých nastavujeme vlastnost PostBackUrl s příslušným id v QueryStringu. Tato vlastnost má za následek, že po kliknutí na tlačítko se nevyvolá PostBack událost na

aktuální stránce, ale na stránce námi definované. Na té pak během životního cyklu stránky, konkrétně v její metodě Page_Load, jejíž ukázka je uvedena ve výpisu 4, kontrolujeme zda se jedná o Cross Page PostBack. Abychom mohli přistupovat k potřebné vlastnosti, musíme na cílové stránce v ASPX části definovat předchozí stránku. Toho docílíme tak, že na začátek vložíme tuto deklaraci <%@ PreviousPageType VirtualPath="PredchoziStranka.aspx" %>.

```
protected void Page_Load(object sender, EventArgs e)
{
       if (PreviousPage != null && PreviousPage.IsCrossPagePostBack)
       {
              if (Request.QueryString["id"] != null)
             {
                    LoadEditControls();
             }
       }
       else
       {
              if (!IsPostBack)
              {
                     Response.Redirect("Questions.aspx");
              }
             else
             {
                    LoadEditControls();
             }
      }
}
```

Výpis 4 Metoda Page_Load editačních stránek

Pokud se nejedná o Cross Page PostBack, je třeba zkontrolovat jestli se nejedná čistě o PostBack už dané editační stránky. K takové situaci dojde, když je uživatel autorizován a vyrenderuje se mu editační stránka, na které provede úpravy a potvrdí je tlačítkem, které vyvolá PostBack událost. Bez této podmínky by se provedené změny neuložily. Pokud se tedy nejedná o PostBack ani Cross Page PostBack je uživatel ihned přesměrován na hlavní stranu a tím jsme mu zabránili ve zneužití cizích dat. Nutno zmínit, že události jako Cross Page PostBack a PostBack nejdou nasimulovat v javascriptu tak, aby se dostaly za náš systém podmínek a jsou tak bezpečným řešením našeho problému editace kategorií, kvízů, oprav a domácích úkolů pouze oprávněnými uživateli.

4.2.8 Serializace XML do webové služby

U API je zajímavé to, že celá WCF služba je vlastně zasílání XML zpráv mezi serverem a konzumentem služby. V našem případě jsme chtěli do těchto zpráv serializovat vygenerované XML pro klienta. A zde podle našeho názoru nastala ta zajímavá věc, že takovou možnost technologie Microsoftu nepodporuje. Narazili jsme na chybu, že XML je pro službu neserializovatelné. Proto jsme tuto situaci museli vyřešit tak, že celé vygenerované XML převádíme do Stringu, který už serializovatelný je. Zároveň se zaručujeme, že takový řetězec je validní XML a je tak možné jej nahrát zpět do XML, tudíž konzument může na své straně

použít metodu pro načtení XML z textového řetězce.

4.2.9 Nastavení App.config

Další zajímavostí při implementaci webové služby, tedy přesněji knihovny pro její konzumaci a desktopové aplikace, která ji využívá bylo nastavení config souboru. Celá zajímavost spočívá v tom, že jsme předpokládali, že pokud nastavíme využívání webové služby v App.config v knihovně, kterou pak budeme referencovat v desktopové aplikaci, že bude vše fungovat. Opak byl pravdou. Proto, abychom mohli tuto knihovnu využít, musí mít aplikace nastaveny stejné parametry ohledně služby jako jsou v referencované knihovně. V našem případě, kdy máme zdrojové kódy knihovny dostupné se nejedná o žádný problém, ale pokud bychom knihovnu měli dostupnou pouze v DLL mohlo by dojít k zajímavé situaci. Zmiňované nastavení je zobrazeno ve výpisu 5.

```
<system.serviceModel>
	<bindings>
		<wsHttpBinding>
			<binding name="WSHttpBinding_IQuizerWCFService">
				<reliableSession enabled="true" />
				<security mode="None" />
			</binding>
		</wsHttpBinding>
	</bindings>
	<client>
	<endpoint
	address="http://158.196.98.22:30384/QuizerWCFService.svc/QuizerWCFService"
	binding="wsHttpBinding"
	bindingConfiguration="WSHttpBinding_IQuizerWCFService"
	contract="QuizerServiceNew.IQuizerWCFService"
	name="WSHttpBinding_IQuizerWCFService" />
	</client>
</system.serviceModel>
```

Výpis 5. Nastavení služby v App.config

4.2.10 Notifikační systém

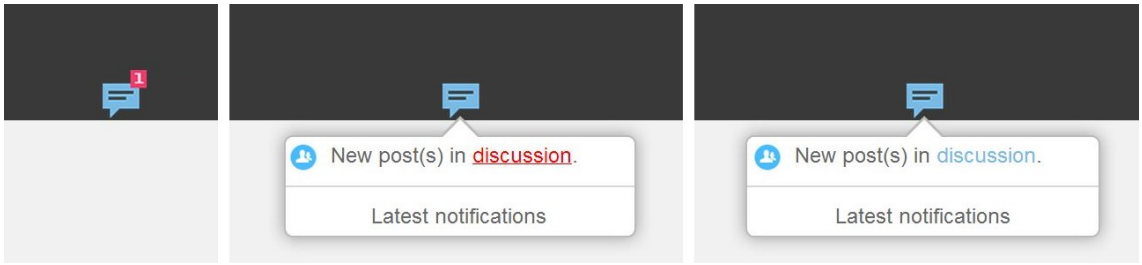
Notifikační systém je založen na technologiích SignalR a SqlDependency. První zmíněná se stará o zasílání upozornění ze serveru na klienta v reálném čase a druhá slouží pro zachytávání událostí na základě změny výsledku hlídaného SQL dotazu v databázi. Princip je blíže popsán v analýze a návrhu. Pro fungující použití SqlDependency musí dotaz splňovat předpoklady, které jsou zmíněny zde [8]. Při implementaci jsme se nevyhli problémům spojeným s nedodržením těchto pravidel.

Všechny metody zajišťující správné fungování musí být na straně klienta implementovány v javasriptu nebo pomocí jQuery. Na straně serveru pak máme Hub. Pomocí něho pak na základě změny v databázi posíláme zprávy přes SignalR. Pro každý druh notifikace jsme vytvořili její šablonu ve formě HTML. Na notifikace na straně klienta je uživatel

upozorňován zvášlním tlačítkem vlevo nahoře ve webové aplikaci. Nové upozornění je signalizováno změnou jeho vzhledu. Některé ze změn si můžeme prohlédnout na obrázku 4.24.

Obrázek 4.24 Některé z typů upozornění

Po kliknutí na tlačítko se zobrazí panel s upozorněními, kde je každé upozornění doprovázeno odkazem na stránku s obsahem, na který je uživatel upozorňován. Také zavádíme rozdíl mezi novými upozorněními a starými. Postup zobrazení upozornění až k jeho statusu starého upozornění je znázorněn na obrázku 4.25.



Obrázek 4.25 Postup zobrazení nového upozornění

Při implementaci fungovalo vše bez problému dokud jsme neotestovali situaci, kdy měl uživatel otevřených více oken aplikace zároveň (pro každé okno vlastní connectionId v SignalR). Tehdy docházelo k nestandardním situacím, kdy byl uživatel upozorňován na své akce nebo se akce prováděly stále dokola mezi otevřenými okny. Tuto situaci jsme vyřešili naimplementováním vlastního managera, který se stará o všechna připojení podle uživatele. Takovýto manager bylo potřeba napsat pro každý implementovaný Hub. V našem případě tedy pro Notifikace, Diskuze a Kategorie. Díky němu máme v každém okamžiku přehled o všech připojeních uživatele v daném kontextu a můžeme tak k nim přistupovat jinak než k ostatním. Například když posíláme HTML příspěvek v diskuzi pošleme na všechna připojení uživatele příspěvek s tlačítkem pro možnost jeho smazání, kdežto všem ostatním pošleme jeho základní verzi. Když jsme implementovali více Hubů nastal problém s jejich spouštěním, které je umístěno v hlavní Master Page. Ukázalo se však, že řešení je jednoduché. Pro spuštění všech Hubů nesmíme spouštět každý zvlášť. Děje se tak pro všechny automaticky při volání `$.connection.hub.start()` .

Další problém nastal když jsme přešli na jinou stránku a klikli na tlačítko upozornění. Otevřený panel byl prázdný. Důvod byl jasný. Jelikož jsme všehna upozornění vkládali skrze javasript na straně klienta, existovali jen tam. Proto jsme vytvořili systém pro udržování jejich historie. Data manageru této historie, manageru notifikací a managerů připojení SignalR klientů řešíme pomocí slovníkové databáze Redis. Použití Redisu mělo i další důvod. Kromě našeho vlastního managera SignalR klientů totiž existuje zároveň i ten, který je součástí SignalR. Jeho rozšíření je možné právě prostřednictvím Redisu jako tzv. backplane, který umožní komunikaci mezi klienty připojenými přes různé servery. V našem prostředí disponujeme pouze jedním serverem, jedná se tak o zabezpečení případného rozšíření. V současné době však toto rozšíření SignalR není k dispozici z nám neznámých důvodů, i když mnoho návodů tento postup popisuje. Proto jsme se na serveru StackOverflow.com zeptali na důvod. Na tuto otázku [10] nám odpověděl i sám David Fowler jeden ze zakladatelů SignalR. Dozvěděli jsme se však pouze to, že tento balíček bude dostupný až v další verzi 1.1, která by mohla být dostupná pravděpodobně do několika týdnů. Výsledkem tedy je, že Redis používáme jen ve všech našich managerech a máme připravené vše potřebné pro rozšíření SignalR na tuto databázi.

4.2.11 Změna hierarchie kategorií

V naší aplikaci počítáme s tím, že funkcionalitu změny hierarchie kategorií bude používat pouze uživatel v roli admin a pouze ojediněle. Přesto jsme se soustředili i na uzamčení XML souboru při jeho editaci. To proto, aby nevznikla situace kdy dva admin uživatelé upravují stejný soubor a mohlo by tak dojít ke ztrátě aktualizace nebo špinavému čtení, řečeno databázovou terminologií. Vzhledem k tomu, že systém drag and drop, pro který využíváme díky použití komponenty třetí strany jménem ASTreeView [18], integrujeme tak, že každá jedinečná změna provedená v režimu editace (zapnutý systém drag and drop) se ihned ukládá. To znamená, že v případě, kdy uživatelům umožňujeme čtení během editace, tak se v tomto případě nejedná o špinavé čtení.

Celou tuto funkcionalitu řešíme pomocí třídy FileStream, které v konstruktoru nastavujeme cestu k souboru a vlastnosti FileMode, FileAccess a FileShare. Tedy v případě čtení nastavíme FileAccess.Read a FileShare.ReadWrite, což znamená, že soubor chceme pouze číst a proto nastavujeme sdílení mezi procesy pro čtení i zápis. V případě zápisu jde o FileAccess.Write a FileShare.Read. Tímto vytvoříme zámek pro zápis aktuálnímu procesu, ale ostaním umožníme čtení. Při implementaci došlo k problémům ohledně funkčnosti zámku. Nastavení FileStreamů pro čtení i zápis bylo korektní, ale přesto nebylo funkční. Chyba při čtení, která hlásila, že soubor je používán jiným procesem byla vzhledem k nastavení sdílení mezi procesy ukazatelem, že nastavení tohoto sdílení řešíme na špatné úrovni aplikace. Řešením nastalé situace proto bylo přesunutí xml souboru a FileStreamu pro zápis do Global.asax. V případě, že jsme používali FileStream pro čtení, stačilo jej použít lokálně. Ovšem pro FileStream pro zápis se musí přistupovat v global.asax.

4.3 Nasazení

Prostředí pro nasazení naší webové aplikace společně s hostovanou službou API je realizováno na virtuálním stroji dostupném na školním serveru. Zde jsme měli předinstalován systém Windows Server 2008 R2 Datacenter. Jedinou nevýhodou, která pramení i z předintalovaného systému je, že námi používaná technologie SignalR nebude dostupná pro typ přenosu s využitím WebSockets. Po úvodních problémech s nedostatkem místa na disku jsme mohli přistoupit k instalaci IIS, Microsoft SQL Serveru a dalších potřebných součástí.

4.3.1 Instalace a konfigurace databáze Redis

Pro instalaci databáze Redis jsme museli využít jeho MS Open Tech portu [11] pro Windows. Jeho nevýhodou může být, že není aktuální s unixovým Redisem, který je momentálně ve verzi 2.6 na rozdíl od jeho portu, který je pro Redis ve verzi 2.4. Abychom mohli Redis používat bez jakéhokoli manuálního spouštění serveru museli jsme jej nainstalovat jako Windows službu. Tu je však po instalaci potřeba napoprvé spustit manuálně. Poté už je její spouštění automatické. Zprvu jsme vše dělali ručně přes příkazovou řádku. Později jsme však narazili na zjednodušující instalační balíček [12] autora Rui Lopes.

Po instalaci přišla na řadu konfigurace. Pro nastavení přihlašování k Redis databázi bylo pro její bezpečnost potřeba v konfiguračním souboru změnit nastavení používaného portu, který je v základním nastavení nastaven na 6379. My jsme jej nastavili na 8805. Jeho změnou snižujeme riziko napadení. Dále bylo také důležité vytvořit alespoň 32 znaků dlouhé heslo, které je součástí connection stringu. To celé proto, že případný utočník může testovat až 150 tisíc hesel za sekundu jak je zmíněno v konfiguračním souboru.

4.3.2 Nastavení FTP, Portů

Pro WCF webovou službu, která slouží jako API pro přístup do databáze našeho systému jsme museli podle [17] nejprve v nastavení Bindings na IIS v příslušném webu přidat nový port a pak na Windows Serveru povolit pro tento port přístup skrze firewall, který je definován ve webconfigu naší aplikace. Konkrétně jde o port 30384. Další port s číslem 1443 jsme museli povolit pro dostupnost SQL Serveru a také povolit v předešlé kapitole zmíněný port 8805 pro Redis.

Pro snadnější a pohodlnější práci při nasazování nové verze webové aplikace do produkčního prostředí jsme museli doinstalovat FTP Server funkcionalitu na dosavadní instalaci IIS 7.0. Poté jsme přidali novou FTP webSite a na firewallu povolili port 21. Pro přenos na FTP používáme klienta FileMozzila. Po zadání IP adresy, loginu a hesla jsme se úspěšně přihlásili na server, ale pak nastal problém. Ten spočíval v tom, že jsme nebyli schopni listovat cílovou directory. Řešení problému bylo přenastavit typ připojení klienta na active. Poté již vše fungovalo správně a my jsme mohli používat ftp přenos.

Kapitola 5

5. Závěr

Naším úkolem bylo naimplementovat sociální síť pro tvorbu kvízových otázek spolu s API poskytujícím data a vyhodnocovací funkce aplikacím třetích stran. Webová aplikace měla umožňovat vytváření kvízových otázek spolu s jejich členěním do kategorií, které měly tvořit editovatelnou hierarchii. Při vytváření otázek pak měla existovat podpora jejich tvorby hlavně pro oblast databázové terminologie a SQL dotazů. Dalším požadavkem byla existence prvků moderní sociální sítě fungující v reálném čase jako je upozorňovací systém nebo možnost diskuze. Dalším požadavkem bylo, aby uživatel mohl navrhnout opravu otázky.

Po chybách v průběhu implementace aplikací, po změně databáze a ztraceném čase při snaze o vlastní řešení asistenta při tvorbě kvízů s využitím sémantických dat se s trochou nadsázky dá říct, že jsme celý systém implementovali dvakrát. I přes tato úskalí se nám však díky snaze o co nejlepší výsledek povedlo vytvořit sociální síť pro tvorbu kvízových otázek s požadovanou funkčností, která někdy jde i nad rámec oficiálního zadání diplomové práce. Podařilo se nám také vytvořit desktopovou aplikaci jako ukázku využití našeho API, které je úspěšně použito v jedné z bakalářských prací. Díky oborovému zaměření sociální sítě na oblast databází byl vytvořen nový druh odpovědi pro SQL dotazy s možností jejich spouštění nad reálným schématem a také systém pro generování podobných SQL dotazů, což považujeme za jedny z hlavních přínosů práce. V jejím rámci jsme také pracovali s novou technologií SignalR pro zajištění funkčních požadavků na získávání dat v reálném čase. Podle potenciálu této technologie změnit v příštích letech vývoj webových aplikací pomocí technologie ASP.NET se jedná díky jejímu použití o další z přínosů našeho systému. Některé části technologie jsou však ještě v raných fázích vývoje, což se projevilo například ve snaze o rozšiřitelnost SignalR na Redis.

Toto otevírá potenciální možnosti dalšího vývoje systému. Ty vidíme ve zmíněné rozšiřitelnosti SignalR na Redis a také přepsání více funkcí do real-time režimu. Dále pro potřebu snížení zátěže serveru a pro jeho rozšiřitelnost můžeme celý systém rozdělit na více serverů (Redis server, SQL server, aplikační server). Dalšími možnostmi může být vytvoření dalších typů možných odpovědí nebo možnost jejich sdílení mezi otázkami. Nabízí se také vylepšení podpory tvorby kvízových otázek například ve zapracování parametrizace otázek na základě [1] nebo vyladění fyzického návrhu SQL databáze.

Kapitola 6

6. Literatura

[1] Jeffrey D. Ullman. Gradiance On-Line Accelerated Learning Guide for Authors. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science - Volume* 38, s. 3 - 6. ACSC, 2005.

[2] *.NET Framework*. Microsoft corporation. *MSDN* [online]. 2012 [cit. 11.4.2013]. Dostupné z: http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2(v=vs.100).aspx

[3] Christina Unger , Lorenz Bühmann , Jens Lehmann , Axel-Cyrille Ngonga Ngomo , Daniel Gerber , Philipp Cimiano. Template-based question answering over RDF data, In *Proceedings of the 21st international conference on World Wide Web*, April 16-20, 2012, Lyon, France

[4] Andreea-Georgiana Zbranca, Diana Andreea Gorea, Lucian Bentea. *Qedia - Natural Language Queries on DBPedia* [online] 2010 [cit. datum]. Dostupné z: http://www.slideshare.net/lucianb/qedia-natural-language-queries-on-dbpedia

[5] *ADO.NET Architecture.* Microsoft corporation. *MSDN* [online]. 2.8.2012 [cit. 11.4.2013]. Dostupné z: http://msdn.microsoft.com/en-us/library/27y4ybxw(v=vs.110).aspx

[6] *SKOS Core Guide.* W3C [online]. 2.11.2005 [cit. 11.4.2013]. Dostupné z: http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102/

[7] DBpedia [online]. 28.3.2013 [cit. 11.4.2013]. Dostupné z: http://dbpedia.org/About

[8] *Special Considerations Using Query Notification (ADO.NET).* Microsoft corporation. *MSDN* [online]. 2009 [cit. 11.4.2013]. Dostupné z: http://msdn.microsoft.com/en-us/library/aewzkxxh.aspx

[9] Redis [online]. [cit. 11.4.2013]. Dostupné z: http://redis.io/documentation

[10] *Why is not SignalR.Redis available on NuGet?* StackOverflow [online]. 9.4.2013 [cit. 11.4.2013]. Dostupné z: http://stackoverflow.com/questions/15887195/why-is-not-signalr-redis-available-on-nuget

[11] *Redis on Windows prototype*. Microsoft corporation. [online]. 7.3.2013 [cit. 11.4.2013]. Dostupné z:https://github.com/MSOpenTech/Redis

[12] Rui Lopes. *Redis key-value store (Win32 / Win64 port with Windows service and installer/setup).* [online]. 17.12.2012 [cit. 11.4.2013]. Dostupné z: https://github.com/rgl/redis

[13] ZURBlog. *Super awesome Buttons with CSS3 and RGBA* [online]. 28.4.2012 [cit. 11.4.2013]. Dostupné z: http://www.zurb.com/article/266/super-awesome-buttons-with-css3-and-rgba

[14] DreamLeader. *How to create 3D links rolling on hover* [online]. 22.8.2012 [cit. 11.4.2013]. Dostupné z: http://www.dreamdealer.nl/articles/how_to_create_3d_links_rolling_on_hover.html

[15] Sean Behan. *jQuery PopBox* [online]. 16.5.2012 [cit. 11.4.2013]. Dostupné z: https://github.com/gristmill/jquery-popbox

[16] Suprotim Agarwal. *Create an ASP.NET TextBox Watermark Effect using jQuery* [online]. 12.6.2009 [cit. 11.4.2013]. Dostupné z: http://www.dotnetcurry.com/ShowArticle.aspx?ID=427

[17] *Browsing to a site on port 8080 Windows Server 2008 IIS 7*. ServerFault [online]. 1.2.2010 [cit. 11.4.2013]. Dostupné z: http://serverfault.com/questions/108289/browsing-to-a-site-on-port-8080-windows-server-2008-iis-7

[18] JIN Weijie. *ASTreeView*. *Best free AST.NET TreeView control.* 14.6.2010 [cit. 11.4.2013]. Dostupné z : http://www.astreeview.com/astreeviewdemo/astreeviewdemo1.aspx

A Obsah přiloženého CD

Tato příloha obsahuje popis adresářové struktury přiloženého CD.

Adresář	Popis
/Aplikace	Obsahuje spustitelné soubory aplikací.
/Programátorská příručka	Zde se nachází vývojářská dokumentace všech aplikací.
/Projekt	Adresář s upravitelnými Visual Studio 2012 projekty .
/Text	Text diplomové práce
/Uživatelská příručka	Adresář obsahuje uživatelskou příručku.